Sage Reference Manual: The Sage Notebook

Release 6.3

The Sage Development Team

August 11, 2014

CONTENTS

CHAPTER

ONE

CONFIGURE AND START A NOTEBOOK SERVER

Configure and Start a Notebook Server

The `NotebookObject` is used to configure and launch a Sage Notebook server.

class `sagenb.notebook.notebook_object.`**`NotebookObject`**

Start the Sage Notebook server. More details about using these options, as well as tips and tricks, may be available at this Sage wiki page. If a notebook server is already running in the directory, this will open a browser to the running notebook.

INPUT:

- `directory` – string; directory that contains the Sage notebook files; the default is `.sage/sage_notebook.sagenb`, in your home directory.
- `port` – integer (default: `8080`), port to serve the notebook on.
- `interface` – string (default: `'localhost'`), address of network interface to listen on; give `''` to listen on all interfaces.
- `port_tries` – integer (default: `0`), number of additional ports to try if the first one doesn't work (*not* implemented).
- `secure` – boolean (default: `False`) if True use https so all communication, e.g., logins and passwords, between web browsers and the Sage notebook is encrypted via SSL. You must have OpenSSL installed to use this feature, or if you compile Sage yourself, have the OpenSSL development libraries installed. *Highly recommended!*
- `reset` – boolean (default: `False`) if True allows you to set the admin password. Use this if you forget your admin password.
- `accounts` – boolean (default: `False`) if True, any visitor to the website will be able to create a new account. If False, only the admin can create accounts (currently, this can only be done by running with `accounts=True` and shutting down the server properly (`SIG_INT` or `SIG_TERM`), or on the command line with, e.g.,

```
from sagenb.notebook.notebook import load_notebook
nb = load_notebook("directory_to_run_sage_in")
user_manager = nb.user_manager()
user_manager.set_accounts(True)
user_manager.add_user("username", "password", "email@place", "user")
nb.save()
```

- `automatic_login` – boolean (default: True) whether to pop up a web browser and automatically log into the server as admin. You can override the default browser by setting the `SAGE_BROWSER` environment variable, e.g., by putting

```
export SAGE_BROWSER="firefox"
```

  in the file .bashrc in your home directory.

- `upload` – string (default: None) Full path to a local file (sws, txt, zip) to be uploaded and turned into a worksheet(s). This is equivalent to manually uploading a file via `http://localhost:8080/upload` or to fetching `http://localhost:8080/upload_worksheet?url=file:///...` in a script except that (hopefully) you will already be logged in.

  Warning: If you are running a server for others to log into, set $automatic_login = False$. Otherwise, all of the worksheets on the entire server will be loaded when the server automatically logs into the admin account.

- `timeout` – integer (default: 0) seconds until idle worksheet sessions automatically timeout, i.e., the corresponding Sage session terminates. 0 means "never timeout". If your server is running out of memory, setting a timeout can be useful as this will free the memory used by idle sessions.

- `doc_timeout` – integer (default: 600) seconds until idle live documentation worksheet sessions automatically timeout, i.e., the corresponding Sage session terminates. 0 means "never timeout".

- `server_pool` – list of strings (default: None) list; this option specifies that worksheet processes run as a separate user (chosen from the list in the `server_pool` – see below).

Note: If you have problems with the server certificate hostname not matching, do `notebook.setup()`.

Note: The `require_login` option has been removed. Use $automatic_login$ to control automatic logins instead—$automatic_login = False$ corresponds to $require_login = True$.

EXAMPLES:

1. I just want to run the Sage notebook. Type:

```
notebook()
```

2. I want to run the Sage notebook server on a remote machine and be the only person allowed to log in. Type:

```
notebook(interface='', secure=True)
```

   the first time you do this you'll be prompted to set an administrator password. Use this to login. NOTE: You may have to run `notebook.setup()` again and change the hostname. ANOTHER NOTE: You must have installed pyOpenSSL in order to use secure mode; see the top-level Sage README file or the "Install from Source Code" section in the Sage manual for more information.

3. I want to create a Sage notebook server that is open to anybody in the world to create new accounts. To run the Sage notebook publicly (1) at a minimum run it from a chroot jail or inside a virtual machine (see this Sage wiki page) and (2) use a command like:

```
notebook(interface='', server_pool=['sage1@localhost'],
ulimit='-v 500000', accounts=True, automatic_login=False)
```

The server_pool option specifies that worksheet processes run as a separate user. The ulimit option restricts the memory available to each worksheet processes to 500 MB. See help on the `accounts` option above.

Be sure that `sage_notebook.sagenb/users.pickle` and the contents of `sage_notebook.sagenb/backups` are chmod `og-rwx`, i.e., only readable by the notebook process, since otherwise any user can read `users.pickle`, which contains user email addresses and account information (passwords are stored hashed, so fewer worries there). You will need to use the `directory` option to accomplish this.

INPUT: (more advanced)

- `server_pool` – list of strings (initial default: None), if given, should be a list like ['sage1@localhost', 'sage2@localhost'], where you have setup ssh keys so that typing:

```
ssh sage1@localhost
```

logs in without requiring a password, e.g., by typing `ssh-keygen` as the notebook server user, then putting `~/.ssh/id_rsa.pub` as the file `.ssh/authorized_keys`. Note: you have to get the permissions of files and directories just right – see this Sage wiki page for more details. Also, every user in the server pool must share the same `/tmp` directory right now, so if the machines are separate the server machine must NSF export `/tmp`.

- `server` – string ("twistd" (default) or "flask"). The server to use to server content.
- `profile` – True, False, or file prefix (default: False - no profiling), If True, profiling is saved to a randomly-named file like $sagenb-*-profile*.stats$ in the $DOT_SAGE directory. If a string, that string is used as a prefix for the pstats data file.
- `ulimit` – string (initial default: None – leave as is), if given and `server_pool` is also given, the worksheet processes are run with these constraints. See the ulimit documentation. Common options include:
  - `-t` The maximum amount of cpu time in seconds. NOTE: For Sage, `-t` is the wall time, not cpu time.
  - `-u` The maximum number of processes available to a single user.
  - `-v` The maximum amount of virtual memory available to the process.

  Values are in 1024-byte increments, except for `-t`, which is in seconds, and `-u` which is a positive integer. Example: ulimit="-v 400000 -t 30"

Note: The values of `server_pool` and `ulimit` default to what they were last time the notebook command was called.

OTHER NOTES:

- If you create a file `$DOT_SAGE/notebook.css` then it will get applied when rendering the notebook HTML. This allows notebook administrators to customize the look of the notebook. Note that by default `$DOT_SAGE` is `$HOME/.sage`.

notebook(*directory=None*, *port=8080*, *interface='localhost'*, *port_tries=50*, *secure=False*, *reset=False*, *accounts=None*, *openid=None*, *server_pool=None*, *ulimit=''*, *timeout=None*, *doc_timeout=None*, *upload=None*, *automatic_login=True*, *start_path=''*, *fork=False*, *quiet=False*, *server='twistd'*, *profile=False*, *subnets=None*, *require_login=None*, *open_viewer=None*, *address=None*)

x.__init__(...) initializes x; see help(type(x)) for signature

setup()
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.notebook.notebook_object.**inotebook**(**args*, ***kwds*)
Exactly the same as `notebook(...)` but with `secure=False`.

sagenb.notebook.notebook_object.**test_notebook**(*admin_passwd*, *secure=False*, *directory=None*, *port=8050*, *interface='localhost'*, *verbose=False*)
This function is used to test notebook server functions.

EXAMPLES:

```
sage: from sagenb.notebook.notebook_object import test_notebook
sage: passwd = str(randint(1,1<<128))
sage: nb = test_notebook(passwd, interface='localhost', port=8060)
sage: import urllib
sage: h = urllib.urlopen('http://localhost:8060')
sage: homepage = h.read()
sage: h.close()
sage: 'html' in homepage
True
sage: nb.dispose()
```

CHAPTER

TWO

NOTEBOOK KEYBINDINGS

This module is responsible for setting the keyboard bindings for the notebook.

These are the standard key and mouse bindings available in the notebook:

- *Evaluate Input:* Press **shift-enter**. You can start several calculations at once. If you press **alt-enter** instead, then a new cell is created after the current one. If you press **ctrl-enter** then the cell is split and both pieces are evaluated separately.
- *Tab Completion:* Press **tab** while the cursor is on an identifier. On some web browsers (e.g., Opera) you must use control-space instead of tab.
- *Insert New Cell:* Put the mouse between an output and input until the horizontal line appears and click. If you press Alt-Enter in a cell, the cell is evaluated and a new cell is inserted after it.
- *Delete Cell:* Delete all cell contents, then press **backspace**.
- *Split and Join Cells:* Press **ctrl-;** in a cell to split it into two cells, and **ctrl-backspace** to join them. Press **ctrl-enter** to split a cell and evaluate both pieces.
- *Insert New HTML Cell:* Shift click between cells to create a new HTML cell. Double click on existing HTML to edit it. Use $...$ and $$...$$ to include typeset math in the HTML block.
- *Hide/Show Output:* Click on the left side of output to toggle between hidden, shown with word wrap, and shown without word wrap.
- *Indenting Blocks:* Highlight text and press **>** to indent it all and **<** to unindent it all (works in Safari and Firefox). In Firefox you can also press tab and shift-tab.
- *Comment/Uncomment Blocks:* Highlight text and press **ctrl-.** to comment it and **ctrl-,** to uncomment it. Alternatively, use **ctrl-3** and **ctrl-4**.
- *Parenthesis matching:* To fix unmatched or mis-matched parentheses, braces or brackets, press **ctrl-0**. Parentheses, brackets or braces to the left of or above the cursor will be matched, minding strings and comments. Note, only Python comments are recognized, so this won't work for c-style multiline comments, etc.

CHAPTER

THREE

INTERACT FUNCTIONS IN THE NOTEBOOK

This module implements an `interact()` function decorator for the Sage notebook.

AUTHORS:

- William Stein (2008-03-02): version 1.0 at Sage/Enthought Days 8 in Texas
- Jason Grout (2008-03): discussion and first few prototypes
- Jason Grout (2008-05): `input_grid` control

class `sagenb.notebook.interact.`**`ColorInput`**(*var*, *default_value*, *label=None*, *type=None*, *width=80*, *height=1*, ***kwargs*)

Bases: `sagenb.notebook.interact.InputBox`

An input box `interact()` control.

INPUT:

- `var` - a string; name of variable that this control interacts
- `default_value` - the default value of the variable corresponding to this control
- `label` - a string (default: None); label for this control
- `type` - a type (default: None); the type of this control, e.g., the type 'bool'
- `height` - an integer (default: 1); the number of rows. If greater than 1 a value won't be returned until something outside the textarea is clicked.
- `width` - an integer (default: 80); the character width of this control
- `kwargs` - a dictionary; additional keyword options

EXAMPLES:

```
sage: sagenb.notebook.interact.InputBox('theta', 1, 'theta')
An InputBox interactive control with theta=1 and label 'theta'
sage: sagenb.notebook.interact.InputBox('theta', 1, 'theta', int)
An InputBox interactive control with theta=1 and label 'theta'
```

`render`()

Render this color input box to HTML.

OUTPUT:

- a string - HTML format

EXAMPLES:

```
sage: sagenb.notebook.interact.ColorInput('c', Color('red')).render()
'...table...color...'
```

value_js(*n*)

Return JavaScript that evaluates to value of this control. If n is 0, return code for evaluation by the actual color control. If n is 1, return code for the text area that displays the current color.

INPUT:

- n - integer, either 0 or 1.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.interact.ColorInput('c', Color('red'))
sage: C.value_js(0)
'color'
sage: C.value_js(1)
'this.value'
```

class sagenb.notebook.interact.**InputBox**(*var*, *default_value*, *label=None*, *type=None*, *width=80*, *height=1*, ***kwargs*)

Bases: sagenb.notebook.interact.InteractControl

An input box interact() control.

INPUT:

- var - a string; name of variable that this control interacts
- default_value - the default value of the variable corresponding to this control
- label - a string (default: None); label for this control
- type - a type (default: None); the type of this control, e.g., the type 'bool'
- height - an integer (default: 1); the number of rows. If greater than 1 a value won't be returned until something outside the textarea is clicked.
- width - an integer (default: 80); the character width of this control
- kwargs - a dictionary; additional keyword options

EXAMPLES:

```
sage: sagenb.notebook.interact.InputBox('theta', 1, 'theta')
An InputBox interactive control with theta=1 and label 'theta'
sage: sagenb.notebook.interact.InputBox('theta', 1, 'theta', int)
An InputBox interactive control with theta=1 and label 'theta'
```

render()

Render this control as a string.

OUTPUT:

- a string - HTML format

EXAMPLES:

```
sage: sagenb.notebook.interact.InputBox('theta', 1).render()
'...input...value="1"...theta...'
```

value_js()
Return JavaScript string that will give the value of this control element.

OUTPUT:

- a string - JavaScript

EXAMPLES:

```
sage: sagenb.notebook.interact.InputBox('theta', 1).value_js()
'this.value'
```

class sagenb.notebook.interact.**InputGrid**(*var*, *rows*, *columns*, *default_value=None*, *label=None*, *to_value=<function <lambda> at 0x7fb30a11be60>*, *width=4*)
Bases: sagenb.notebook.interact.InteractControl

A grid interact control.

INPUT:

- `var` - an object; the variable
- `rows` - an integer; the number of rows
- `columns` - an integer; the number of columns
- `default_value` - an object; if this is a scalar, it is put in every cell; if it is a list, it is filled into the cells row by row; if it is a nested list, then it is filled into the cells according to the nesting structure.
- `label` - a string; the label for the control
- `to_value` - a function applied to the nested list from user input when assigning the variable
- `width` - an integer; the width of the input boxes

EXAMPLES:

```
sage: sagenb.notebook.interact.InputGrid('M', 2,2, default_value = 0, label='M')
A 2 x 2 InputGrid interactive control with M=[[0, 0], [0, 0]] and label 'M'
sage: sagenb.notebook.interact.InputGrid('M', 2,2, default_value = [[1,2],[3,4]], label='M')
A 2 x 2 InputGrid interactive control with M=[[1, 2], [3, 4]] and label 'M'
sage: sagenb.notebook.interact.InputGrid('M', 2,2, default_value = [[1,2],[3,4]], label='M', to_
A 2 x 2 InputGrid interactive control with M=[1 2]
[3 4] and label 'M'
sage: sagenb.notebook.interact.InputGrid('M', 1, 3, default_value=[1,2,3], to_value=lambda x: ve
A 1 x 3 InputGrid interactive control with M=(1, 2, 3) and label 'M'
```

render()
Render this control as a string.

OUTPUT:

- string - HTML format

EXAMPLES:

```
sage: sagenb.notebook.interact.InputGrid('M', 1,2).render()
'...table...input...M...'
```

value_js()
Return JavaScript string that will give the value of this control element.

OUTPUT:

- string - JavaScript

EXAMPLES:

```
sage: sagenb.notebook.interact.InputGrid('M', 2,2).value_js()
"...jQuery...table...map...val...join..."
```

class sagenb.notebook.interact.**InteractCanvas**(*controls*, *id*, *layout=None*, *width=None*, ***options*)

Bases: object

Base class for interact() canvases. This is where all the controls along with the output of the interacted function are laid out and rendered.

INPUT:

- controls - a list of InteractControl instances.
- id - an integer or a string; the ID of the cell that contains this InteractCanvas.
- layout - a dictionary with keys 'top','bottom','left','right' and values lists of rows of control variable names. If a dictionary is not passed in, then the value of layout is set to the 'top' value. If None, then all control names are put on separate rows in the 'top' value.
- width - the width of the interact control
- options - any additional keyword arguments (for example, auto_update=False)

EXAMPLES:

```
sage: B = sagenb.notebook.interact.InputBox('x',2)
sage: sagenb.notebook.interact.InteractCanvas([B], 3)
Interactive canvas in cell 3 with 1 controls
```

cell_id()

Return the ID of the cell that contains this interact() control.

OUTPUT:

- an integer or a string

EXAMPLES:

The output below should equal the ID of the current cell:

```
sage: B = sagenb.notebook.interact.InputBox('x',2)
sage: C = sagenb.notebook.interact.InteractCanvas([B], 3); C
Interactive canvas in cell 3 with 1 controls
sage: C.cell_id()
3
```

controls()

Return a list of controls in this canvas.

OUTPUT:

- list of controls

Note: Returns a reference to a mutable list.

EXAMPLES:

```
sage: B = sagenb.notebook.interact.InputBox('x',2)
sage: sagenb.notebook.interact.InteractCanvas([B], 3).controls()
[An InputBox interactive control with x=2 and label 'x']
```

is_auto_update()
Returns True if any change of the values for the controls on this canvas should cause an update. If `auto_update=False` was not specified in the constructor for this canvas, then this will default to True.

OUTPUT:

- a bool

EXAMPLES:

```
sage: B = sagenb.notebook.interact.InputBox('x',2)
sage: canvas = sagenb.notebook.interact.InteractCanvas([B], 3)
sage: canvas.is_auto_update()
True
sage: canvas = sagenb.notebook.interact.InteractCanvas([B], 3, auto_update=False)
sage: canvas.is_auto_update()
False
```

render()
Render in text (HTML) the entire `interact()` canvas.

OUTPUT:

- string - HTML format

EXAMPLES:

```
sage: B = sagenb.notebook.interact.InputBox('x',2)
sage: sagenb.notebook.interact.InteractCanvas([B], 3).render()
'...notruncate...div...interact...table...x...'
```

render_controls(*side='top'*)
Render in text (HTML) form all the input controls.

OUTPUT:

- a string - HTML format

EXAMPLES:

```
sage: B = sagenb.notebook.interact.InputBox('x',2)
sage: sagenb.notebook.interact.InteractCanvas([B], 3).render_controls()
'...table...x...input...2...'
```

render_output()
Render in text (HTML) form the output portion of the `interact()` canvas.

The output contains two special tags, <?TEXT> and <?HTML>, which get replaced at runtime by the text and HTML parts of the output of running the function.

OUTPUT:

- a string - HTML format

EXAMPLES:

```
sage: B = sagenb.notebook.interact.InputBox('x',2)
sage: sagenb.notebook.interact.InteractCanvas([B], 3).render_output()
'...div...interact...3...'
```

wrap_in_outside_frame(*inside*)
Return the entire HTML for the interactive canvas, obtained by wrapping all the inside HTML of the canvas in a div and a table.

INPUT:

- `inside` - a string; HTML

OUTPUT:

- a string - HTML format

EXAMPLES:

```
sage: B = sagenb.notebook.interact.InputBox('x',2)
sage: sagenb.notebook.interact.InteractCanvas([B], 3).wrap_in_outside_frame('<!--inside-->')
'...notruncate...div...interact...table...inside...'
```

class sagenb.notebook.interact.**InteractControl**(*var*, *default_value*, *label=None*)

Bases: sagenb.notebook.interact.InteractElement

Abstract base class for `interact()` controls. These are controls that are used in a specific `interact()`. They have internal state information about the specific function being interacted, etc.

INPUT:

- `var` - a string; name of variable that this control interacts
- `default_value` - the default value of the variable corresponding to this control.
- `label` - a string (default: None); label for this control; if None then defaults to `var`.

EXAMPLES:

```
sage: from sagenb.notebook.interact import InteractControl
sage: InteractControl('x', default_value=5)
A InteractControl (abstract base class)
```

adapt_number()

Return integer index into adapt dictionary of function that is called to adapt the values of this control to Python.

OUTPUT:

- an integer

EXAMPLES:

```
sage: from sagenb.notebook.interact import InteractControl
sage: InteractControl('x', 19/3).adapt_number()       # random -- depends on call order
2
```

cell_id()

Return the ID of the cell that contains this `interact()` control.

OUTPUT:

- an integer or a string

EXAMPLES:

The output below should equal the ID of the current cell:

```
sage: sagenb.notebook.interact.InteractControl('theta', 1).cell_id()
0
```

default_value()

Return the default value of the variable corresponding to this `interact()` control.

OUTPUT:

- an object

EXAMPLES:

```
sage: from sagenb.notebook.interact import InteractControl
sage: InteractControl('x', 19/3).default_value()
19/3
```

html_escaped_default_value()

Returns the HTML escaped default value of the variable corresponding to this interact() control. Note that any HTML that uses quotes around this should use double quotes and not single quotes.

OUTPUT:

- a string

EXAMPLES:

```
sage: from sagenb.notebook.interact import InteractControl
sage: InteractControl('x', '"cool"').html_escaped_default_value()
'&amp;quot;cool&amp;quot;'
sage: InteractControl('x',"'cool'").html_escaped_default_value()
"'cool'"
sage: x = var('x')
sage: InteractControl('x', x^2).html_escaped_default_value()
'x^2'
```

interact(**args*)

Return a string that when evaluated in JavaScript calls the JavaScript interact() function with appropriate inputs for this control.

This method will check to see if there is a canvas attached to this control and whether or not controls should automatically update the output when their values change. If no canvas is associated with this control, then the control will automatically update.

OUTPUT:

- a string - that is meant to be evaluated in JavaScript

EXAMPLES:

```
sage: sagenb.notebook.interact.InteractControl('x', 1).interact()
"...interact...x...1..."
```

label()

Return the text label of this interact() control.

OUTPUT:

- a string

EXAMPLES:

```
sage: from sagenb.notebook.interact import InteractControl
sage: InteractControl('x', default_value=5, label='the x value').label()
'the x value'
```

value_js()

JavaScript that when evaluated gives the current value of this control. This should be redefined in a derived class.

OUTPUT:

- a string - defaults to 'NULL' - this should be redefined.

EXAMPLES:

```
sage: sagenb.notebook.interact.InteractControl('x', default_value=5).value_js()
'NULL'
```

var()

Return the name of the variable that this control interacts.

OUTPUT:

- a string - name of a variable as a string.

EXAMPLES:

```
sage: sagenb.notebook.interact.InteractControl('theta', 1).var()
'theta'
```

class sagenb.notebook.interact.**InteractElement**

Bases: object

x.__init__(...) initializes x; see help(type(x)) for signature

canvas()

Returns the InteractCanvas associated to this element. If no canvas has been set (via the set_canvas() method), then raise a ValueError.

EXAMPLES:

```
sage: from sagenb.notebook.interact import InputBox, InteractCanvas
sage: B = InputBox('x',2)
sage: canvas1 = InteractCanvas([B], 3)
sage: canvas2 = InteractCanvas([B], 3)
sage: B.canvas() is canvas2
True
```

label()

Returns an empty label for this element. This should be overridden for subclasses that need a label.

OUTPUT:

- a string

EXAMPLES:

```
sage: from sagenb.notebook.interact import UpdateButton, InteractElement
sage: b = UpdateButton(1, 'autoupdate')
sage: isinstance(b, InteractElement)
True
sage: b.label()
''
```

set_canvas(*canvas*)

Sets the InteractCanvas on which this element appears. This method is primarily called in the constructor for InteractCanvas.

EXAMPLES:

```
sage: from sagenb.notebook.interact import InputBox, InteractCanvas
sage: B = InputBox('x',2)
sage: canvas1 = InteractCanvas([B], 3)
sage: canvas2 = InteractCanvas([B], 3)
sage: B.canvas() is canvas2
True
sage: B.set_canvas(canvas1)
```

```
sage: B.canvas() is canvas1
True
```

class sagenb.notebook.interact.**JavascriptCodeButton**(*label*, *code*)

Bases: sagenb.notebook.interact.InteractElement

This interact() element displays a button which when clicked executes JavaScript code in the notebook.

EXAMPLES:

```
sage: b = sagenb.notebook.interact.JavascriptCodeButton('Push me', 'alert("2")')
```

render()

Returns the HTML to display this button.

OUTPUT:

- a string - HTML format

EXAMPLES:

```
sage: b = sagenb.notebook.interact.JavascriptCodeButton('Push me', 'alert("2")')
sage: b.render()
'...input...button...Push me...alert("2")...'
```

class sagenb.notebook.interact.**RangeSlider**(*var*, *values*, *default_position*, *label=None*, *display_value=True*)

Bases: sagenb.notebook.interact.SliderGeneric

A range slider interact() control that takes on the given list of values.

INPUT:

- var - a string; name of variable being interacted
- values - a list; a list of the values that the slider will take on
- default_position - an integer 2-tuple; default location that the slider is set to.
- label - a string; alternative label to the left of the slider, instead of the variable.
- display_value - a bool, whether to display the current value below the slider

EXAMPLES:

```
sage: sagenb.notebook.interact.RangeSlider('x', [1..5], (2,3), 'alpha')
Range Slider Interact Control: alpha [1--|3==4|---5]
```

default_position()

Return the default position (as an integer) of the slider.

OUTPUT:

- an integer 2-tuple

EXAMPLES:

```
sage: sagenb.notebook.interact.RangeSlider('x', [1..5], (2,3), 'alpha').default_position()
(2, 3)
```

render()

Render this control as an HTML string.

OUTPUT:

•string - HTML format

EXAMPLES:

```
sage: sagenb.notebook.interact.RangeSlider('x', [1..5], (2,3), 'alpha').render()
'...table...slider...["1","2","3","4","5"]...range...'

sage: sagenb.notebook.interact.RangeSlider('x', [1..5], (2,3), 'alpha', display_value=False)
'...table...slider...null...range...
```

value_js()

Return JavaScript string that will give the value of this control element.

OUTPUT:

•a string - JavaScript

EXAMPLES:

```
sage: sagenb.notebook.interact.RangeSlider('x', [1..5], (2,3), 'alpha').value_js()
"pos[0]+' '+pos[1]"
```

class sagenb.notebook.interact.**Selector**(*var*, *values*, *label=None*, *default=0*, *nrows=None*, *ncols=None*, *width=None*, *buttons=False*)

Bases: sagenb.notebook.interact.InteractControl

A drop down menu or a button bar that when pressed sets a variable to a given value.

INPUT:

•`var` - a string; variable name

•`values` - a list; button values

•`label` - a string (default: None); label off to the left for this button group

•`default` - an integer (default: 0); position of default value in values list.

•`nrows` - an integer (default: None); number of rows

•`ncols` - an integer (default: None); number of columns

•`width` - an integer (default: None); width of all the buttons

•**`buttons` - a bool (default: False); if True use buttons** instead of dropdown

EXAMPLES:

```
sage: sagenb.notebook.interact.Selector('x', [1..5], 'alpha', default=2)
Selector with 5 options for variable 'x'
sage: sagenb.notebook.interact.Selector('x', [1..4], 'alpha', default=2, nrows=2, ncols=2, width
Selector with 4 options for variable 'x'
```

render()

Render this control as a string.

OUTPUT:

•a string - HTML format

EXAMPLES:

```
sage: sagenb.notebook.interact.Selector('x', [1..5]).render()
'...select...x...'
sage: sagenb.notebook.interact.Selector('x', [1..5], buttons=True).render()
'...table...button...x...'
```

use_buttons()
Whether or not to use buttons instead of a drop down menu for this select list.

OUTPUT:

- a bool

EXAMPLES:

```
sage: sagenb.notebook.interact.Selector('x', [1..5]).use_buttons()
False
sage: sagenb.notebook.interact.Selector('x', [1..5], buttons=True).use_buttons()
True
```

value_js()
Return JavaScript string that will give the value of this control element.

OUTPUT:

- a string - JavaScript

EXAMPLES:

```
sage: sagenb.notebook.interact.Selector('x', [1..5]).value_js()
'this.options[this.selectedIndex].value'
sage: sagenb.notebook.interact.Selector('x', [1..5], buttons=True).value_js()
'this.value'
```

class sagenb.notebook.interact.**Slider**(*var*, *values*, *default_position*, *label=None*, *display_value=True*)
Bases: sagenb.notebook.interact.SliderGeneric

A slider interact() control that takes on the given list of values.

INPUT:

- var - a string; name of variable being interacted
- values - a list; a list of the values that the slider will take on
- default_position - an integer; default location that the slider is set to.
- label - a string; alternative label to the left of the slider, instead of the variable.
  - display_value - a bool, whether to display the current value right of the slider

EXAMPLES:

```
sage: sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha')
Slider Interact Control: alpha [1--|3|---5]
```

default_position()
Return the default position (as an integer) of the slider.

OUTPUT:

- an integer

EXAMPLES:

```
sage: sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha').default_position()
2
```

render()
Render this control as an HTML string.

OUTPUT:

- a string - HTML format

EXAMPLES:

```
sage: sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha').render()
'...table...slider...["1","2","3","4","5"]...'

sage: sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha', display_value=False).render()
'...table...slider...null...'
```

value_js()

Return JavaScript string that will give the value of this control element.

OUTPUT:

- a string - JavaScript

EXAMPLES:

```
sage: sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha').value_js()
'position'
```

class sagenb.notebook.interact.**SliderGeneric**(*var*, *values*, *default_value*, *label=None*, *display_value=True*)

Bases: sagenb.notebook.interact.InteractControl

An abstract slider interact() control that takes on the given list of values.

INPUT:

- var - a string; name of variable being interacted
- **values - a list; a list of the values that the slider** will take on
- default_value - an object; default value of the slider.
- label - a string; alternative label to the left of the slider, instead of the variable.
- display_value - a bool; whether to display the current value on the slider

EXAMPLES:

```
sage: sagenb.notebook.interact.SliderGeneric('x', [1..5], 2, 'alpha')
Abstract Slider Interact Control: alpha [1--|2|---5]
```

display_value()

Returns whether to display the value on the slider.

OUTPUT:

- a bool

EXAMPLES:

```
sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha').display_value()
True
```

values()

Return list of values the slider acts on.

OUTPUT:

- a list

EXAMPLES:

```
sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha').values()
[1, 2, 3, 4, 5]
```

values_js()
Returns JavaScript array representation of values or 'null' if display_value=False

OUTPUT:

- a string

EXAMPLES:

```
sage: sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha').values_js()
'["1","2","3","4","5"]'
sage: sagenb.notebook.interact.Slider('x', [1..5], 2, 'alpha', False).values_js()
'null'
sage: sagenb.notebook.interact.Slider('x', [pi..2*pi], 2, 'alpha').values_js()
'["pi","pi + 1","pi + 2","pi + 3"]'
```

class sagenb.notebook.interact.**TextControl**(*var*, *data*)
Bases: sagenb.notebook.interact.InteractControl

A text field interact() control

INPUT:

- data - a string; the HTML value of the text field

EXAMPLES:

```
sage: sagenb.notebook.interact.TextControl('x', 'something')
Text Interact Control: something
```

render()
Render this control as an HTML string.

OUTPUT:

- a string - HTML format

EXAMPLES:

```
sage: sagenb.notebook.interact.TextControl('x', 'something').render()
'...div...something...'
```

class sagenb.notebook.interact.**UpdateButton**(*cell_id*, *var*)
Bases: sagenb.notebook.interact.JavascriptCodeButton

Creates an interact() button element. A click on the button triggers recomputation of the cell with the current values of the variables.

INPUT:

- cell_id - an integer or string; the ambient cell's ID
- var ` - a variable which is used in the layout

EXAMPLES:

```
sage: b = sagenb.notebook.interact.UpdateButton(0, 'auto_update')
sage: b.render()
'...input...button...Update...0...'
```

var()
Return the name of the variable that this control interacts.

OUTPUT:

- a string - name of a variable as a string.

EXAMPLES:

```
sage: sagenb.notebook.interact.UpdateButton(1, 'auto_update').var()
'auto_update'
```

sagenb.notebook.interact.**automatic_control**(*default*)
Automagically determine the type of control from the default value of the variable.

INPUT:

- `default` - the default value for `v` given by the function; see the documentation to `interact()` for details.

OUTPUT:

- an `interact()` control

EXAMPLES:

```
sage: sagenb.notebook.interact.automatic_control('')
Interact input box labeled None with default value ''
sage: sagenb.notebook.interact.automatic_control(15)
Interact input box labeled None with default value 15
sage: sagenb.notebook.interact.automatic_control(('start', 15))
Interact input box labeled 'start' with default value 15
sage: sagenb.notebook.interact.automatic_control((1,250))
Slider: None [1.0--|1.0|---250.0]
sage: sagenb.notebook.interact.automatic_control(('alpha', (1,250)))
Slider: alpha [1.0--|1.0|---250.0]
sage: sagenb.notebook.interact.automatic_control((2,(0,250)))
Slider: None [0.0--|2.00400801603|---250.0]
sage: sagenb.notebook.interact.automatic_control(('alpha label', (2,(0,250))))
Slider: alpha label [0.0--|2.00400801603|---250.0]
sage: sagenb.notebook.interact.automatic_control((2, ('alpha label',(0,250))))
Slider: alpha label [0.0--|2.00400801603|---250.0]
sage: C = sagenb.notebook.interact.automatic_control((1,52, 5)); C
Slider: None [1--|1|---52]
sage: C.values()
[1, 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 52]
sage: sagenb.notebook.interact.automatic_control((17, (1,100,5)))
Slider: None [1--|16|---100]
sage: sagenb.notebook.interact.automatic_control([1..4])
Button bar with 4 buttons
sage: sagenb.notebook.interact.automatic_control([1..100])
Drop down menu with 100 options
sage: sagenb.notebook.interact.automatic_control((1..100))
Slider: None [1--|1|---100]
sage: sagenb.notebook.interact.automatic_control((5, (1..100)))
Slider: None [1--|5|---100]
sage: sagenb.notebook.interact.automatic_control(matrix(2,2))
Interact 2 x 2 input grid control labeled None with default value [0, 0, 0, 0]
```

class sagenb.notebook.interact.**checkbox**(*default=True*, *label=None*)
Bases: `sagenb.notebook.interact.input_box`

A checkbox interactive control. Use this in conjunction with the `interact()` command.

INPUT:

- `default` - a bool (default: True); whether box should be checked or not
- `label` - a string (default: None) text label rendered to the left of the box

EXAMPLES:

```
sage: checkbox(False, "Points")
Interact checkbox labeled 'Points' with default value False
sage: checkbox(True, "Points")
Interact checkbox labeled 'Points' with default value True
sage: checkbox(True)
Interact checkbox labeled None with default value True
sage: checkbox()
Interact checkbox labeled None with default value True
```

class `sagenb.notebook.interact.`**`color_selector`**(*default=(0, 0, 1)*, *label=None*, *widget='jpicker'*, *hide_box=False*)

Bases: `sagenb.notebook.interact.input_box`

A color selector (also called a color chooser, picker, or tool) interactive control. Use this with the `interact()` command.

INPUT:

- `default` - an instance of or valid constructor argument to `Color` (default: (0,0,1)); the selector's default color; a string argument must be a valid color name (e.g., 'red') or HTML hex color (e.g., '#abcdef')
- `label` - a string (default: None); the label rendered to the left of the selector.
- `widget` - a string (default: 'jpicker'); the color selector widget to use; choices are 'colorpicker', 'jpicker' and 'farbtastic'
- `hide_box` - a boolean (default: False); whether to hide the input box associated with the color selector widget

EXAMPLES:

```
sage: color_selector()
Interact color selector labeled None, with default RGB color (0.0, 0.0, 1.0), widget 'jpicker',
sage: color_selector((0.5, 0.5, 1.0), widget='jpicker')
Interact color selector labeled None, with default RGB color (0.5, 0.5, 1.0), widget 'jpicker',
sage: color_selector(default = Color(0, 0.5, 0.25))
Interact color selector labeled None, with default RGB color (0.0, 0.5, 0.25), widget 'jpicker',
sage: color_selector('purple', widget = 'colorpicker')
Interact color selector labeled None, with default RGB color (0.50..., 0.0, 0.50...), widget 'cc
sage: color_selector('crayon', widget = 'colorpicker')
Traceback (most recent call last):
...
ValueError: unknown color 'crayon'
sage: color_selector('#abcdef', label='height', widget='jpicker')
Interact color selector labeled 'height', with default RGB color (0.6..., 0.8..., 0.9...), widge
sage: color_selector('abcdef', label='height', widget='jpicker')
Traceback (most recent call last):
...
ValueError: unknown color 'abcdef'
```

hide_box()

Return whether to hide the input box associated with this color selector.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: color_selector().hide_box()
False
sage: color_selector('green', hide_box=True, widget='jpicker').hide_box()
True
sage: color_selector((0.75,0.5,0.25)).hide_box()
False
```

widget()

Return the name of the HTML widget for this color selector.

OUTPUT:

- a string; the widget's name

EXAMPLES:

```
sage: color_selector().widget()
'jpicker'
sage: color_selector('#abcdef', hide_box=True, widget='farbtastic').widget()
'farbtastic'
sage: color_selector(widget='colorpicker').widget()
'colorpicker'
sage: color_selector(default=Color(0,0.5,0.25)).widget()
'jpicker'
```

class sagenb.notebook.interact.**control**(*label=None*)

An interactive control object used with the `interact()` command. This is the abstract base class.

INPUTS:

- `label` - a string

EXAMPLES:

```
sage: sagenb.notebook.interact.control('a control')
Interative control 'a control' (abstract base class)
```

label()

Return the label of this control.

OUTPUT:

- a string

EXAMPLES:

```
sage: sagenb.notebook.interact.control('a control').label()
'a control'
sage: selector([1,2,7], 'alpha').label()
'alpha'
```

set_label(*label*)

Set the label of this control.

INPUT:

- `label` - a string

EXAMPLES:

```
sage: C = sagenb.notebook.interact.control('a control')
sage: C.set_label('sage'); C
Interative control 'sage' (abstract base class)
```

sagenb.notebook.interact.**html**(*s*)

Print the input string `s` in a form that tells the notebook to display it in the HTML portion of the output. This function has no return value.

INPUT:

- `s` - a string

EXAMPLES:

```
sage: sagenb.notebook.interact.html('hello')
<html>hello</html>
```

sagenb.notebook.interact.**html_color_selector**(*id*, *change*, *input_change*, *default='000000'*, *widget='jpicker'*, *hide_box=False*)

Return HTML representation of a jQuery color selector.

INPUT:

- `id` - an integer or string; an identifier (e.g., cell ID) for this selector
- `change` - a string; JavaScript code to execute when the color selector changes.
- `default` - a string (default: `'000000'`); default color as a 6-character HTML hexadecimal string.
- `widget` - a string (default: 'jpicker'); the color selector widget to use; choices are 'colorpicker', 'jpicker' and 'farbtastic'
- `hide_box` - a boolean (default: False); whether to hide the input box associated with the color selector widget

OUTPUT:

- a string - HTML that creates the slider.

EXAMPLES:

```
sage: sagenb.notebook.interact.html_color_selector(0, 'alert("changed")', '', default='0afcac')
'...<table>...jpicker...'
sage: sagenb.notebook.interact.html_color_selector(99, 'console.log(color);', '', default='fedcb
'...<table>...colorpicker...'
```

sagenb.notebook.interact.**html_rangeslider**(*id*, *values*, *callback*, *steps*, *default_l=0*, *default_r=1*, *margin=0*)

Return the HTML representation of a jQuery range slider.

INPUT:

- `id` - a string; the DOM ID of the slider (better be unique)
- `values` - a string; 'null' or JavaScript string containing array of values on slider
- `callback` - a string; JavaScript that is executed whenever the slider is done moving
- `steps` - an integer; number of steps from minimum to maximum value
- `default_l` - an integer (default: 0); the default position of the left edge of the slider
- `default_r` - an integer (default: 1); the default position of the right edge of the slider

- margin - an integer (default: 0); size of margin to insert around the slider

OUTPUT:

- a string - HTML format

EXAMPLES:

We create a jQuery range slider. If you do the following in the notebook you should obtain a slider that when moved pops up a window showing its current position:

```
sage: from sagenb.notebook.interact import html_rangeslider, html
sage: html(html_rangeslider('slider-007', 'null', 'alert(pos[0]+", "+pos[1])', steps=5, default_
<html>...slider...range...</html>
```

sagenb.notebook.interact.**html_slider**(*id*, *values*, *callback*, *steps*, *default=0*, *margin=0*)

Return the HTML representation of a jQuery slider.

INPUT:

- id - a string; the DOM ID of the slider (better be unique)
- values - a string; 'null' or JavaScript string containing array of values on slider
- callback - a string; JavaScript that is executed whenever the slider is done moving
- steps - an integer; number of steps from minimum to maximum value
- default - an integer (default: 0); the default position of the slider
- margin - an integer (default: 0); size of margin to insert around the slider

OUTPUT:

- a string - HTML format

EXAMPLES:

We create a jQuery HTML slider. If you do the following in the notebook you should obtain a slider that when moved pops up a window showing its current position:

```
sage: from sagenb.notebook.interact import html_slider, html
sage: html(html_slider('slider-007', 'null', 'alert(position)', steps=5, default=2, margin=5))
<html>...slider...</html>
```

class sagenb.notebook.interact.**input_box**(*default=None*, *label=None*, *type=None*, *width=80*, *height=1*, ***kwargs*)

Bases: sagenb.notebook.interact.control

An input box interactive control. Use this in conjunction with the interact() command.

INPUT:

- default - an object; the default put in this input box
- label - a string; the label rendered to the left of the box.
- type - a type; coerce inputs to this; this doesn't have to be an actual type, since anything callable will do.
- height - an integer (default: 1); the number of rows. If greater than 1 a value won't be returned until something outside the textarea is clicked.
- width - an integer; width of text box in characters
- kwargs - a dictionary; additional keyword options

EXAMPLES:

```
sage: input_box("2+2", 'expression')
Interact input box labeled 'expression' with default value '2+2'
sage: input_box('sage', label="Enter your name", type=str)
Interact input box labeled 'Enter your name' with default value 'sage'
sage: input_box('Multiline\nInput',label='Click to change value',type=str,height=5)
Interact input box labeled 'Click to change value' with default value 'Multiline\nInput'
```

default()

Return the default value of this input box.

OUTPUT:

- an object

EXAMPLES:

```
sage: input_box('2+2', 'Expression').default()
'2+2'
sage: input_box(x^2 + 1, 'Expression').default()
x^2 + 1
sage: checkbox(True, "Points").default()
True
```

render(*var*)

Return rendering of this input box as an `InputBox` to be used for an `interact()` canvas. Basically this specializes this input to be used for a specific function and variable.

INPUT:

- `var` - a string (variable; one of the variable names input to `f`)

OUTPUT:

- an `InputBox` instance

EXAMPLES:

```
sage: input_box("2+2", 'Exp').render('x')
An InputBox interactive control with x='2+2' and label 'Exp'
```

type()

Return the type that elements of this input box are coerced to or None if they are not coerced (they have whatever type they evaluate to).

OUTPUT:

- a type

EXAMPLES:

```
sage: input_box("2+2", 'expression', type=int).type()
<type 'int'>
sage: input_box("2+2", 'expression').type() is None
True
```

class sagenb.notebook.interact.**input_grid**(*nrows*, *ncols*, *default=None*, *label=None*, *to_value=<function <lambda> at 0x7fb30a123cf8>*, *width=4*)

Bases: `sagenb.notebook.interact.control`

An input grid interactive control. Use this in conjunction with the `interact()` command.

INPUT:

- `nrows` - an integer
- `ncols` - an integer
- `default` - an object; the default put in this input box
- `label` - a string; the label rendered to the left of the box.
- `to_value` - a list; the grid output (list of rows) is sent through this function. This may reformat the data or coerce the type.
- `width` - an integer; size of each input box in characters

NOTEBOOK EXAMPLE:

```
@interact
def _(m = input_grid(2,2, default = [[1,7],[3,4]],
                     label='M=', to_value=matrix),
      v = input_grid(2,1, default=[1,2],
                     label='v=', to_value=matrix)):
    try:
        x = m\v
        html('$$%s %s = %s$$'%(latex(m), latex(x), latex(v)))
    except:
        html('There is no solution to $$%s x=%s$$'%(latex(m), latex(v)))
```

EXAMPLES:

```
sage: input_grid(2,2, default = 0, label='M')
Interact 2 x 2 input grid control labeled M with default value 0
sage: input_grid(2,2, default = [[1,2],[3,4]], label='M')
Interact 2 x 2 input grid control labeled M with default value [[1, 2], [3, 4]]
sage: input_grid(2,2, default = [[1,2],[3,4]], label='M', to_value=MatrixSpace(ZZ,2,2))
Interact 2 x 2 input grid control labeled M with default value [[1, 2], [3, 4]]
sage: input_grid(1, 3, default=[[1,2,3]], to_value=lambda x: vector(flatten(x)))
Interact 1 x 3 input grid control labeled None with default value [[1, 2, 3]]
```

default()

Return the default value of this input grid.

OUTPUT:

- an object

EXAMPLES:

```
sage: input_grid(2,2, default=1).default()
1
```

render(*var*)

Return rendering of this input grid as an `InputGrid` to be used for an `interact()` canvas. Basically this specializes this input to be used for a specific function and variable.

INPUT:

- `var` - a string (variable; one of the variable names input to `f`)

OUTPUT:

- an `InputGrid` instance.

EXAMPLES:

```
sage: input_grid(2,2).render('x')
A 2 x 2 InputGrid interactive control with x=[[None, None], [None, None]] and label 'x'
```

sagenb.notebook.interact.**interact**(*f*, *layout=None*, *width='800px'*)

Use interact as a decorator to create interactive Sage notebook cells with sliders, text boxes, radio buttons, check boxes, and color selectors. Simply put `@interact` on the line before a function definition in a cell by itself, and choose appropriate defaults for the variable names to determine the types of controls (see tables below).

INPUT:

- `f` - a Python function
- `layout` (optional) - a dictionary with keys 'top', 'bottom', 'left', 'right' and values lists of rows of control variable names. Controls are laid out according to this pattern. If `layout` is not a dictionary, it is assumed to be the 'top' value. If `layout` is None, then all controls are assigned separate rows in the `top` value.

EXAMPLES:

In each example below we use a single underscore for the function name. You can use *any* name you want; it does not have to be an underscore.

We create an interact control with two inputs, a text input for the variable `a` and a `y` slider that runs through the range of integers from 0 to 19.

```
sage: @interact
... def _(a=5, y=(0..20)): print a + y
...
<html>...

sage: @interact(layout=[['a','b'],['d']])
... def _(a=x^2, b=(0..20), c=100, d=x+1): print a+b+c+d
...
<html>...

sage: @interact(layout={'top': [['a', 'b']], 'left': [['c']], 'bottom': [['d']]})
... def _(a=x^2, b=(0..20), c=100, d=x+1): print a+b+c+d
...
<html>...
```

Draw a plot interacting with the "continuous" variable `a`. By default continuous variables have exactly 50 possibilities.

```
sage: @interact
... def _(a=(0,2)):
...     show(plot(sin(x*(1+a*x)), (x,0,6)), figsize=4)
<html>...
```

Interact a variable in steps of 1 (we also use an unnamed function):

```
sage: @interact
... def _(n=(10,100,1)):
...     show(factor(x^n - 1))
<html>...
```

Interact two variables:

```
sage: @interact
... def _(a=(1,4), b=(0,10)):
...     show(plot(sin(a*x+b), (x,0,6)), figsize=3)
<html>...
```

Place a block of text among the controls:

```
sage: @interact
... def _(t1=text_control("Factors an integer."), n="1"):
```

```
...       print factor(Integer(n))
<html>...
```

You do not have to use interact as a decorators; you can also simply write `interact(f)` where `f` is any Python function that you have defined, though this is frowned upon. E.g., `f` can also be a library function as long as it is written in Python:

```
sage: interact(matrix)    # put ZZ, 2,2,[1..4] in boxes...
<html>...
```

If your the time to evaluate your function takes awhile, you may not want to have it reevaluated every time the inputs change. In order to prevent this, you can add a keyword `auto_update=False` to your function to prevent it from updating whenever the values are changed. This will cause a button labeled 'Update' to appear which you can click on to re-evaluate your function.

```
sage: @interact
... def _(n=(10,100,1), auto_update=False):
...       show(factor(x^n - 1))
<html>...
```

DEFAULTS:

Defaults for the variables of the input function determine interactive controls. The standard controls are `input_box`, `slider`, `range_slider`, `checkbox`, `selector`, `input_grid`, and `color_selector`. There is also a text control (see the defaults below).

- `u = input_box(default=None, label=None, type=None)` - input box with given `default`; use `type=str` to get input as an arbitrary string
- `u = slider(vmin, vmax=None, step_size=1, default=None, label=None)` - slider with given list of possible values; `vmin` can be a list
- `u = range_slider(vmin, vmax=None, step_size=1, default=None, label=None)` - range slider with given list of possible values; `vmin` can be a list
- `u = checkbox(default=True, label=None)` - a checkbox
- `u = selector(values, label=None, nrows=None, ncols=None, buttons=False)` - a dropdown menu or buttons (get buttons if `nrows`, `ncols`, or `buttons` is set, otherwise a dropdown menu)
- `u = input_grid(nrows, ncols, default=None, label=None, to_value=lambda x:x, width=4)` - an editable grid of objects (a matrix or array)
- `u = color_selector(default=(0,0,1), label=None, widget='farbtastic', hide_box=False)` - a color selector with a possibly hidden input box; the `widget` can also be `'jpicker'` or `'colorpicker'`
- `u = text_control(value='')` - a block of text

You can also create a color selector by setting the default value for an `input_box` to `Color(...)`.

There are also some convenient defaults that allow you to make controls automatically without having to explicitly specify them. E.g., you can make `x` a continuous slider of values between `u` and `v` by just writing `x=(u,v)` in the argument list of your function. These are all just convenient shortcuts for creating the controls listed above.

- `u` - blank input_box field
- `u = element` - input_box with `default=element`, if element not below.
- `u = (umin,umax)` - continuous slider (really 100 steps)

- `u = (umin,umax,du)` - slider with step size `du`
- `u = list` - buttons if `len(list)` at most 5; otherwise, drop down
- `u = generator` - a slider (up to 10000 steps)
- `u = bool` - a checkbox
- `u = Color('blue')` - a color selector; returns `Color` object
- `u = (default, v)` - `v` as above, with given `default` value
- `u = (label, v)` - `v` as above, with given `label` (a string)
- `u = matrix` - an `input_grid` with `to_value` set to `matrix.parent()` and default values given by the matrix

---

Note: Suppose you would like to make an interactive with a default RGB color of `(1,0,0)`, so the function would have signature `f(color=(1,0,0))`. Unfortunately, the above shortcuts reinterpret the `(1,0,0)` as a discrete slider with step size 0 between 1 and 0. Instead you should do the following:

```
sage: @interact
... def _(v = input_box((1,0,0))):
...       show(plot(sin,color=v))
<html>...
```

An alternative:

```
sage: @interact
... def _(c = color_selector((1, 0, 0))):
...       show(plot(sin, color = c))
<html>...
```

MORE EXAMPLES:

We give an input box that allows one to enter completely arbitrary strings:

```
sage: @interact
... def _(a=input_box('sage', label="Enter your name", type=str)):
...       print "Hello there %s"%a.capitalize()
<html>...
```

The scope of variables that you control via `interact()` are local to the scope of the function being interacted with. However, by using the `global` Python keyword, you can still modify global variables as follows:

```
sage: xyz = 10
sage: @interact
... def _(a=('xyz',5)):
...       global xyz
...       xyz = a
<html>...
```

If you enter the above you obtain an `interact()` canvas. Entering values in the box changes the global variable `xyz`. Here's a example with several controls:

```
sage: @interact
... def _(title=["A Plot Demo", "Something silly", "something tricky"], a=input_box(sin(x*sin(x*
...     clr = Color('red'), thickness=[1..30], zoom=(1,0.95,..,0.1), plot_points=(200..2000)):
...     html('<h1 align=center>%s</h1>'%title)
...     print plot_points
...     show(plot(a, -zoom*pi,zoom*pi, color=clr, thickness=thickness, plot_points=plot_points))
<html>...
```

For a more compact color control, use an empty label, a different widget ('colorpicker' or 'jpicker'), and hide the input box:

```
sage: @interact
... def _(color=color_selector((1,0,1), label='', widget='colorpicker', hide_box=True)):
...     show(plot(x/(8/7+sin(x)), (x,-50,50), fill=True, fillcolor=color))
<html>...
```

We give defaults and name the variables:

```
sage: @interact
... def _(a=('first', (1,4)), b=(0,10)):
...       show(plot(sin(a*x+sin(b*x)), (x,0,6)), figsize=3)
<html>...
```

Another example involving labels, defaults, and the slider command:

```
sage: @interact
... def _(a = slider(1, 4, default=2, label='Multiplier'),
...       b = slider(0, 10, default=0, label='Phase Variable')):
...     show(plot(sin(a*x+b), (x,0,6)), figsize=4)
<html>...
```

An example where the range slider control is useful:

```
sage: @interact
... def _(b = range_slider(-20, 20, 1, default=(-19,3), label='Range')):
...     plot(sin(x)/x, b[0], b[1]).show(xmin=b[0],xmax=b[1])
<html>...
```

An example using checkboxes, obtained by making the default values bools:

```
sage: @interact
... def _(axes=('Show axes', True), square=False):
...       show(plot(sin, -5,5), axes=axes, aspect_ratio = (1 if square else None))
<html>...
```

An example generating a random walk that uses a checkbox control to determine whether points are placed at each step:

```
sage: @interact
... def foo(pts = checkbox(True, "points"), n = (50,(10..100))):
...       s = 0; v = [(0,0)]
...       for i in range(n):
...            s += random() - 0.5
...            v.append((i, s))
...       L = line(v, rgbcolor='#4a8de2')
...       if pts: L += points(v, pointsize=20, rgbcolor='black')
...       show(L)
<html>...
```

You can rotate and zoom into 3-D graphics while interacting with a variable:

```
sage: @interact
... def _(a=(0,1)):
...     x,y = var('x,y')
...     show(plot3d(sin(x*cos(y*a)), (x,0,5), (y,0,5)), figsize=4)
<html>...
```

A random polygon:

```
sage: pts = [(random(), random()) for _ in xrange(20)]
sage: @interact
... def _(n = (4..len(pts)), c=Color('purple') ):
...     G = points(pts[:n],pointsize=60) + polygon(pts[:n], rgbcolor=c)
...     show(G, figsize=5, xmin=0, ymin=0)
<html>...
```

Two "sinks" displayed simultaneously via a contour plot and a 3-D interactive plot:

```
sage: @interact
... def _(q1=(-1,(-3,3)), q2=(-2,(-3,3))):
...     x,y = var('x,y')
...     f = q1/sqrt((x+1)^2 + y^2) + q2/sqrt((x-1)^2+(y+0.5)^2)
...     C = contour_plot(f, (-2,2), (-2,2), plot_points=30, contours=15, cmap='cool')
...     show(C, figsize=3, aspect_ratio=1)
...     show(plot3d(f, (x,-2,2), (y,-2,2)), figsize=4)
<html>...
```

This is similar to above, but you can select the color map from a dropdown menu:

```
sage: @interact
... def _(q1=(-1,(-3,3)), q2=(-2,(-3,3)),
...    cmap=['autumn', 'bone', 'cool', 'copper', 'gray', 'hot', 'hsv',
...          'jet', 'pink', 'prism', 'spring', 'summer', 'winter']):
...     x,y = var('x,y')
...     f = q1/sqrt((x+1)^2 + y^2) + q2/sqrt((x-1)^2+(y+0.5)^2)
...     C = contour_plot(f, (x,-2,2), (y,-2,2), plot_points=30, contours=15, cmap=cmap)
...     show(C, figsize=3, aspect_ratio=1)
<html>...
```

A quadratic roots etch-a-sketch:

```
sage: v = []
sage: html('<h2>Quadratic Root Etch-a-sketch</h2>')
<html>...<h2>Quadratic Root Etch-a-sketch</h2>...</html>
sage: @interact
... def _(a=[-10..10], b=[-10..10], c=[-10..10]):
...       f = a*x^2 + b*x + c == 0; show(f)
...       soln = solve(a*x^2 + b*x + c == 0, x)[0].rhs()
...       show(soln)
...       P = tuple(CDF(soln))
...       v.append(P)
...       show(line(v, rgbcolor='purple') + point(P, pointsize=200))
<html>...
```

In the following example, we only generate data for a given `n` once, so that as one varies `p` the data does not randomly change. We do this by simply caching the results for each `n` in a dictionary.:

```
sage: data = {}
sage: @interact
... def _(n=(500,(100,5000,1)), p=(1,(0.1,10))):
...     n = int(n)
...     if not data.has_key(n):
...         data[n] = [(random(), random()) for _ in xrange(n)]
...     show(points([(x^p,y^p) for x,y in data[n]], rgbcolor='black'), xmin=0, ymin=0, axes=Fals
<html>...
```

A conchoid:

```
sage: @interact
... def _(k=(1.2,(1.1,2)), k_2=(1.2,(1.1,2)), a=(1.5,(1.1,2))):
...     u, v = var('u,v')
...     f = (k^u*(1+cos(v))*cos(u), k^u*(1+cos(v))*sin(u), k^u*sin(v)-a*k_2^u)
...     show(parametric_plot3d(f, (u,0,6*pi), (v,0,2*pi), plot_points=[40,40], texture=(0,0.5,0)
<html>...
```

An input grid:

```
sage: @interact
... def _(A=matrix(QQ,3,3,range(9)), v=matrix(QQ,3,1,range(3))):
...     try:
...         x = A\v
...         html('$$%s %s = %s$$'%(latex(A), latex(x), latex(v)))
...     except:
...         html('There is no solution to $$%s x=%s$$'%(latex(A), latex(v)))
<html>...
```

sagenb.notebook.interact.**list_of_first_n**(*v*, *n*)

Given an iterator v, return first n elements it produces as a list.

INPUT:

- v - an iterator
- n - an integer

OUTPUT:

- a list

EXAMPLES:

```
sage: from itertools import takewhile
sage: p100 = takewhile(lambda x: x < 100, Primes())
sage: sagenb.notebook.interact.list_of_first_n(p100, 10)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
sage: sagenb.notebook.interact.list_of_first_n((1..5), 10)
[1, 2, 3, 4, 5]
sage: sagenb.notebook.interact.list_of_first_n(QQ, 10)
[0, 1, -1, 1/2, -1/2, 2, -2, 1/3, -1/3, 3]
```

sagenb.notebook.interact.**new_adapt_number**()

Return an integer, always counting up, and starting with 0. This is used for saving the adapt methods for controls. An adapt method is just a function that coerces data into some object, e.g., makes sure the control always produces int's.

OUTPUT:

- an integer

EXAMPLES:

```
sage: sagenb.notebook.interact.new_adapt_number()    # random output -- depends on when called
1
```

class sagenb.notebook.interact.**range_slider**(*vmin*, *vmax=None*, *step_size=None*, *default=None*, *label=None*, *display_value=True*)

Bases: sagenb.notebook.interact.slider_generic

An interactive range slider control, which can be used in conjunction with the interact() command.

INPUT:

- vmin - an object
- vmax - object or None; if None then vmin must be a list, and the slider then varies over elements of the list.
- step_size - integer (default: 1)
- default - a 2-tuple of objects (default: None); default range is "closest" in vmin or range to this default.
- label - a string
- display_value - a bool, whether to display the current value below the slider

EXAMPLES:

We specify both vmin and vmax. We make the default $(3, 4)$ but since neither is one of $3/17$-th spaced values between 2 and 5, the closest values: $52/17$ and $67/17$, are instead chosen as the default:

```
sage: range_slider(2, 5, 3/17, (3,4), 'alpha')
Range Slider: alpha [2--|52/17==67/17|---5]
```

Here we give a list:

```
sage: range_slider([1..10], None, None, (3,7), 'alpha')
Range Slider: alpha [1--|3==7|---10]
```

default_index()

Return default index into the list of values.

OUTPUT:

- an integer 2-tuple

EXAMPLES:

```
sage: range_slider(2, 5, 1/2, (3,4), 'alpha').default_index()
(2, 4)
```

render(*var*)

Render the interact() control for the given function and variable.

INPUT:

- var - string; variable name

OUTPUT:

- a RangeSlider instance

EXAMPLES:

```
sage: S = range_slider(0, 10, 1, default=(3,7), label='theta'); S
Range Slider: theta [0--|3==7|---10]
sage: S.render('x')
Range Slider Interact Control: theta [0--|3==7|---10]
sage: range_slider(2, 5, 2/7, (3,4), 'alpha').render('x')
Range Slider Interact Control: alpha [2--|20/7==4|---5]
```

sagenb.notebook.interact.**recompute**(*cell_id*)

Evaluates the interact() function associated to the cell cell_id. This typically gets called after a call to update().

INPUT:

- cell_id - a string or an integer; the ID of an interact() cell

EXAMPLES:

The following outputs __SAGE_INTERACT_RESTART__ to indicate that not all the state of the interact() canvas has been set up yet (this setup happens when JavaScript calls certain functions):

```
sage: sagenb.notebook.interact.recompute(10)
__SAGE_INTERACT_RESTART__
```

sagenb.notebook.interact.**reset_state**()

Reset the interact() state of this sage process.

EXAMPLES:

```
sage: sagenb.notebook.interact.state  # random output
{1: {'function': <function g at 0x72aaab0>, 'variables': {'m': 3, 'n': 5}, 'adapt': {1: <bound m
sage: from sagenb.notebook.interact import reset_state
sage: reset_state()
sage: sagenb.notebook.interact.state
{}
```

class sagenb.notebook.interact.**selector**(*values*, *label=None*, *default=None*, *nrows=None*, *ncols=None*, *width=None*, *buttons=False*)

Bases: sagenb.notebook.interact.control

A drop down menu or a button bar that when pressed sets a variable to a given value. Use this in conjunction with the interact() command.

We use the same command to create either a drop down menu or selector bar of buttons, since conceptually the two controls do exactly the same thing - they only look different. If either nrows or ncols is given, then you get a buttons instead of a drop down menu.

INPUT:

- values - [val0, val1, val2, ...] or [(val0, lbl0), (val1,lbl1), ...] where all labels must be given or given as None.
- label - a string (default: None); if given, this label is placed to the left of the entire button group
- default - an object (default: 0); default value in values list
- nrows - an integer (default: None); if given determines the number of rows of buttons; if given buttons option below is set to True
- ncols - an integer (default: None); if given determines the number of columns of buttons; if given buttons option below is set to True
- width - an integer (default: None); if given, all buttons are the same width, equal to this in HTML ex units's.
- buttons - a bool (default: False); if True, use buttons

EXAMPLES:

```
sage: selector([1..5])
Drop down menu with 5 options
sage: selector([1,2,7], default=2)
Drop down menu with 3 options
sage: selector([1,2,7], nrows=2)
Button bar with 3 buttons
sage: selector([1,2,7], ncols=2)
Button bar with 3 buttons
sage: selector([1,2,7], width=10)
Drop down menu with 3 options
```

```
sage: selector([1,2,7], buttons=True)
Button bar with 3 buttons
```

We create an `interact()` that involves computing charpolys of matrices over various rings:

```
sage: @interact
... def _(R=selector([ZZ,QQ,GF(17),RDF,RR]), n=(1..10)):
...       M = random_matrix(R, n)
...       show(M)
...       show(matrix_plot(M,cmap='Oranges'))
...       f = M.charpoly()
...       print f
<html>...
```

Here we create a drop-down:

```
sage: @interact
... def _(a=selector([(2,'second'), (3,'third')])):
...       print a
<html>...
```

default()

Return the default choice for this control.

OUTPUT:

- an integer, with 0 corresponding to the first choice.

EXAMPLES:

```
sage: selector([1,2,7], default=2).default()
1
```

render(*var*)

Return rendering of this button as a `Selector` instance to be used for an `interact()` canvas.

INPUT:

- `var` - a string (variable; one of the variable names input to `f`)

OUTPUT:

- a `Selector` instance

EXAMPLES:

```
sage: selector([1..5]).render('alpha')
Selector with 5 options for variable 'alpha'
```

values()

Return the list of values or (val, lbl) pairs that this selector can take on.

OUTPUT:

- a list

EXAMPLES:

```
sage: selector([1..5]).values()
[1, 2, 3, 4, 5]
sage: selector([(5,'fifth'), (8,'eight')]).values()
[(5, 'fifth'), (8, 'eight')]
```

class sagenb.notebook.interact.**slider**(*vmin*, *vmax=None*, *step_size=None*, *default=None*, *label=None*, *display_value=True*)

Bases: sagenb.notebook.interact.slider_generic

An interactive slider control, which can be used in conjunction with the interact() command.

INPUT:

- vmin - an object
- vmax - an object (default: None); if None then vmin must be a list, and the slider then varies over elements of the list.
- step_size - an integer (default: 1)
- default - an object (default: None); default value is "closest" in vmin or range to this default.
- label - a string
- display_value - a bool, whether to display the current value to the right of the slider

EXAMPLES:

We specify both vmin and vmax. We make the default 3, but since 3 isn't one of $3/17$-th spaced values between 2 and 5, $52/17$ is instead chosen as the default (it is closest):

```
sage: slider(2, 5, 3/17, 3, 'alpha')
Slider: alpha [2--|52/17|---5]
```

Here we give a list:

```
sage: slider([1..10], None, None, 3, 'alpha')
Slider: alpha [1--|3|---10]
```

The elements of the list can be anything:

```
sage: slider([1, 'x', 'abc', 2/3], None, None, 'x', 'alpha')
Slider: alpha [1--|x|---2/3]
```

default_index()

Return default index into the list of values.

OUTPUT:

- an integer

EXAMPLES:

```
sage: slider(2, 5, 1/2, 3, 'alpha').default_index()
2
```

render(*var*)

Render the interact() control for the given function and variable.

INPUT:

- var - a string; variable name

OUTPUT:

- a Slider instance

EXAMPLES:

```
sage: S = slider(0, 10, 1, default=3, label='theta'); S
Slider: theta [0--|3|---10]
sage: S.render('x')
```

```
Slider Interact Control: theta [0--|3|---10]
sage: slider(2, 5, 2/7, 3, 'alpha').render('x')
Slider Interact Control: alpha [2--|20/7|---5]
```

class sagenb.notebook.interact.**slider_generic**(*vmin*, *vmax=None*, *step_size=None*, *label=None*, *display_value=True*)

Bases: sagenb.notebook.interact.control

display_value()

Returns whether to display the value on the slider.

OUTPUT:

- a bool

EXAMPLES:

```
sagenb.notebook.interact.slider_generic(1,10,1/2).display_value()
True
```

values()

Returns list of values that this slider takes on, in order.

OUTPUT:

- a list

Note: This is a reference to a mutable list.

EXAMPLES:

```
sage: sagenb.notebook.interact.slider(1,10,1/2).values()
[1, 3/2, 2, 5/2, 3, 7/2, 4, 9/2, 5, 11/2, 6, 13/2, 7, 15/2, 8, 17/2, 9, 19/2, 10]
```

class sagenb.notebook.interact.**text_control**(*value=''*)

Bases: sagenb.notebook.interact.control

Text that can be inserted among other interact() controls.

INPUT:

- value - HTML for the control

EXAMPLES:

```
sage: text_control('something')
Text field: something
```

render(*var*)

Return rendering of the text field

INPUT:

- var - a string (variable; one of the variable names input to f)

OUTPUT:

- a TextControl instance

sagenb.notebook.interact.**update**(*cell_id*, *var*, *adapt*, *value*, *globs*)

Called when updating the positions of an interactive control. Note that this just causes the values of the variables to be updated; it does not reevaluate the function with the new values.

INPUT:

- `cell_id` - an integer or string; the ID of an `interact()` cell
- `var` - an object; a variable associated to that cell
- `adapt` - in integer; the number of the adapt function
- `value` - an object; new value of the control
- `globs` - global variables.

EXAMPLES:

The following outputs __SAGE_INTERACT_RESTART__ to indicate that not all the state of the `interact()` canvas has been set up yet (this setup happens when JavaScript calls certain functions):

```
sage: sagenb.notebook.interact.update(0, 'a', 0, '5', globals())
__SAGE_INTERACT_RESTART__
```

CHAPTER

FOUR

A CELL

A cell is a single input/output block. Worksheets are built out of a list of cells.

class sagenb.notebook.cell.**Cell**(*id*, *input*, *out*, *worksheet*)

Bases: sagenb.notebook.cell.Cell_generic

Creates a new compute cell.

INPUT:

- id - an integer or string; the new cell's ID
- input - a string; this cell's input
- out - a string; this cell's output
- worksheet - a sagenb.notebook.worksheet.Worksheet instance; this cell's worksheet object

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C == loads(dumps(C))
True
```

cell_output_type()

Returns this compute cell's output type.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.cell_output_type()
'wrap'
sage: C.set_cell_output_type('nowrap')
sage: C.cell_output_type()
'nowrap'
```

changed_input_text()

Returns the changed input text for this compute cell, deleting any previously stored text.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: initial_version=C.version()
sage: C.changed_input_text()
''
sage: C.set_changed_input_text('3+3')
sage: C.input_text()
u'3+3'
sage: C.changed_input_text()
u'3+3'
sage: C.changed_input_text()
''
sage: C.version()-initial_version
0
```

cleaned_input_text()

Returns this compute cell's "cleaned" input text, i.e., its input with all of its percent directives removed. If this cell is interacting, it returns the interacting text.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '%hide\n%maxima\n2+3', '5', None)
sage: C.cleaned_input_text()
u'2+3'
```

computing()

Returns whether this compute cell is queued for evaluation by its worksheet object.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = W.new_cell_after(0, "2^2")
sage: C.computing()
False
sage: nb.delete()
```

delete_files()

Deletes all of the files associated with this compute cell.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, 'plot(sin(x),0,5)', '', W)
sage: C.evaluate()
sage: W.check_comp(wait=9999)      # random output -- depends on computer speed
('d', Cell 0: in=plot(sin(x),0,5), out=
<html><font color='black'><img src='cell://sage0.png'></font></html>

)
sage: C.files()      # random output -- depends on computer speed
['sage0.png']
```

```
sage: C.delete_files()
sage: C.files()
[]
sage: W.quit()
sage: nb.delete()
```

delete_output()

Deletes all output in this compute cell. This also deletes the files, since they appear as output of the cell.

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None); C
Cell 0: in=2+3, out=5
sage: C.delete_output()
sage: C
Cell 0: in=2+3, out=
```

When output is deleted, any files in the cell directory are deleted as well:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save('{{{\nplot(sin(x),(x,0,5))\n///\n20\n}}}')
sage: C = W.cell_list()[0]
sage: C.evaluate()
sage: W.check_comp(wait=9999)      # random output -- depends on computer speed
('d', Cell 0; in=plot(sin(x),(x,0,5)), out=
<html><font color='black'><img src='cell://sage0.png'></font></html>

)
sage: C.files()      # random output -- depends on computer speed
['sage0.png']
sage: C.delete_output()
sage: C.files()
[]
sage: W.quit()
sage: nb.delete()
```

directory()

Returns the name of this compute cell's directory, creating it, if it doesn't already exist.

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.directory()
'.../home/sage/0/cells/0'
sage: nb.delete()
```

edit_text(*ncols=0*, *prompts=False*, *max_out=None*)

Returns the text displayed for this compute cell in the Edit window.

INPUT:

- `ncols` - an integer (default: 0); the number of word wrap columns

- `prompts` - a boolean (default: False); whether to strip interpreter prompts from the beginning of each line
- `max_out` - an integer (default: None); the maximum number of characters to return

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.edit_text()
u'{{{id=0|\n2+3\n///\n5\n}}}'
sage: C = sagenb.notebook.cell.Cell(0, 'ěščřžýáíéd'Ď', 'ěščřžýáíéd'Ď', None)
sage: C.edit_text()
u'{{{id=0|\něščřž\xfd\xe1\xed\xe9d'Ď\n///\něščřž\xfd\xe1\xed\xe9d'Ď\n}}}'
```

evaluate(*introspect=False*, *time=None*, *username=None*)

Evaluates this compute cell.

INPUT:

- `introspect` - a pair [`before_cursor`, `after_cursor`] of strings (default: False)
- `time` - a boolean (default: None); whether to return the time the computation takes
- `username` - a string (default: None); name of user doing the evaluation

EXAMPLES:

We create a notebook, worksheet, and cell and evaluate it in order to compute 3^5:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save('{{{\n3^5\n}}}')
sage: C = W.cell_list()[0]; C
Cell 0: in=3^5, out=
sage: C.evaluate(username='sage')
sage: W.check_comp(wait=9999)      # random output -- depends on computer speed
('d', Cell 0: in=3^5, out=
243
)
sage: C      # random output -- depends on computer speed
Cell 0: in=3^5, out=
243
sage: W.quit()
sage: nb.delete()
```

evaluated()

Returns whether this compute cell has been successfully evaluated in a currently running session. This is not about whether the output of the cell is valid given the input.

OUTPUT:

- a boolean

EXAMPLES: We create a worksheet with a cell that has wrong output:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save('{{{\n2+3\n///\n20\n}}}')
```

```
sage: C = W.cell_list()[0]
sage: C
Cell 0: in=2+3, out=
20
```

We re-evaluate that input cell:

```
sage: C.evaluate()
sage: W.check_comp(wait=9999)      # random output -- depends on computer speed
('w', Cell 0: in=2+3, out=)
```

Now the output is right:

```
sage: C      # random output -- depends on computer speed
Cell 0: in=2+3, out=
```

And the cell is considered to have been evaluated.

```
sage: C.evaluated()      # random output -- depends on computer speed
True

sage: W.quit()
sage: nb.delete()
```

files()

Returns a list of all the files in this compute cell's directory.

OUTPUT:

- a list of strings

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, 'plot(sin(x),0,5)', '', W)
sage: C.evaluate()
sage: W.check_comp(wait=9999)      # random output -- depends on computer speed
('d', Cell 0: in=plot(sin(x),0,5), out=
<html><font color='black'><img src='cell://sage0.png'></font></html>

)
sage: C.files()      # random output -- depends on computer speed
['sage0.png']
sage: W.quit()
sage: nb.delete()
```

files_html(*out*)

Returns HTML to display the files in this compute cell's directory.

INPUT:

- `out` - a string; files to exclude. To exclude bar, foo, ..., use the format `'cell://bar cell://foo ...'`

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, 'plot(sin(x),0,5)', '', W)
sage: C.evaluate()
sage: W.check_comp(wait=9999)      # random output -- depends on computer speed
('d', Cell 0: in=plot(sin(x),0,5), out=
<html><font color='black'><img src='cell://sage0.png'></font></html>

)
sage: C.files_html('')      # random output -- depends on computer speed
'<img src="/home/sage/0/cells/0/sage0.png?...">'
sage: W.quit()
sage: nb.delete()
```

has_output()

Returns whether this compute cell has any output.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.has_output()
True
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '', None)
sage: C.has_output()
False
```

html(*wrap=None*, *div_wrap=True*, *do_print=False*, *publish=False*)

Returns the HTML for this compute cell.

INPUT:

- `wrap` - an integer (default: None); the number of word wrap columns
- `div_wrap` - a boolean (default: True); whether to wrap the output in outer div elements
- `do_print` - a boolean (default: False); whether to return output suitable for printing
- `publish` - a boolean (default: False); whether to render a published cell

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.html()
u'...cell_outer_0...2+3...5...'
```

input_text()

Returns this compute cell's input text.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.input_text()
u'2+3'
```

interrupt()

Sets this compute cell's evaluation as interrupted.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = W.new_cell_after(0, "2^2")
sage: C.interrupt()
sage: C.interrupted()
True
sage: C.evaluated()
False
sage: nb.delete()
```

interrupted()

Returns whether this compute cell's evaluation has been interrupted.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = W.new_cell_after(0, "2^2")
sage: C.interrupt()
sage: C.interrupted()
True
sage: nb.delete()
```

introspect()

Returns compute cell's introspection text.

OUTPUT:

- a string 2-tuple ("before" and "after" text) or boolean (not introspecting)

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, 'sage?', '', W)
sage: C.introspect()
False
sage: C.evaluate(username='sage')
sage: W.check_comp(9999)      # random output -- depends on computer speed
('d', Cell 0: in=sage?, out=)
sage: C.introspect()
[u'sage?', '']
sage: W.quit()
sage: nb.delete()
```

introspect_html()
Returns this compute cell's introspection text, setting it to '', if none is available.

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, 'sage?', '', W)
sage: C.introspect()
False
sage: C.evaluate(username='sage')
sage: W.check_comp(9999)      # random output -- depends on computer speed
('d', Cell 0: in=sage?, out=)
sage: C.introspect_html()     # random output -- depends on computer speed
u'...<div class="docstring">...sage...</pre></div>...'
sage: W.quit()
sage: nb.delete()
```

is_asap()
Returns whether this compute cell is to be evaluated as soon as possible (ASAP).

OUTPUT:

- a boolean

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.is_asap()
False
sage: C.set_asap(True)
sage: C.is_asap()
True
```

is_auto_cell()
Returns whether this compute cell is evaluated automatically when its worksheet object starts up.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.is_auto_cell()
False
sage: C = sagenb.notebook.cell.Cell(0, '#auto\n2+3', '5', None)
sage: C.is_auto_cell()
True
```

is_html()
Returns whether this is an HTML compute cell, e.g., its system is 'html'. This is typically specified by the percent directive `%html`.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, "%html\nTest HTML", None, None)
sage: C.system()
u'html'
sage: C.is_html()
True
sage: C = sagenb.notebook.cell.Cell(0, "Test HTML", None, None)
sage: C.is_html()
False
```

is_interacting()

Returns whether this compute cell is currently sagenb.notebook.interact.interact()ing.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = W.new_cell_after(0, "@interact\ndef f(a=slider(0,10,1,5):\n    print a^2")
sage: C.is_interacting()
False
```

is_interactive_cell()

Returns whether this compute cell contains sagenb.notebook.interact.interact() either as a function call or decorator.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = W.new_cell_after(0, "@interact\ndef f(a=slider(0,10,1,5):\n    print a^2")
sage: C.is_interactive_cell()
True
sage: C = W.new_cell_after(C.id(), "2+2")
sage: C.is_interactive_cell()
False
sage: nb.delete()
```

is_no_output()

Returns whether this is a "no output" compute cell, i.e., we don't care about its output.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.is_no_output()
False
sage: C.set_no_output(True)
sage: C.is_no_output()
True
```

next_compute_id()
Returns the ID of the next compute cell in this compute cell's worksheet object. If this cell is *not* in the worksheet, it returns the ID of the worksheet's *first* compute cell. If this *is* the last compute cell, it returns its *own* ID.

OUTPUT:

- an integer or string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save('foo\n{{{\n2+3\n///\n5\n}}}bar\n{{{\n2+8\n///\n10\n}}}')
sage: W.new_cell_after(1, "2^2")
Cell 4: in=2^2, out=
sage: [W.get_cell_with_id(i).next_compute_id() for i in [1, 4, 3]]
[4, 3, 3]
```

output_html()
Returns this compute cell's HTML output.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.output_html()
''
sage: C.set_output_text('5', '<strong>5</strong>')
sage: C.output_html()
u'<strong>5</strong>'
```

output_text(*ncols=0*, *html=True*, *raw=False*, *allow_interact=True*)
Returns this compute cell's output text.

INPUT:

- `ncols` - an integer (default: 0); the number of word wrap columns
- `html` - a boolean (default: True); whether to output HTML
- `raw` - a boolean (default: False); whether to output raw text (takes precedence over HTML)
- `allow_interact` - a boolean (default: True); whether to allow `sagenb.notebook.interact.interact()`ion

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.output_text()
u'<pre class="shrunk">5</pre>'
sage: C.output_text(html=False)
u'<pre class="shrunk">5</pre>'
sage: C.output_text(raw=True)
```

```
u'5'
sage: C = sagenb.notebook.cell.Cell(0, 'ěščřžýáíéd'Ď', 'ěščřžýáíéd'Ď', W)
sage: C.output_text()
u'<pre class="shrunk">ěščřž\xfd\xe1\xed\xe9d'Ď</pre>'
sage: C.output_text(raw=True)
u'ěščřž\xfd\xe1\xed\xe9d'Ď'
```

parse_html(*s*, *ncols*)

Parses HTML for output, escaping and wrapping HTML and removing script elements.

INPUT:

- `s` - a string; the HTML to parse
- `ncols` - an integer; the number of word wrap columns

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.parse_html('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">\n<html><head></head><
'&lt;!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0...Test</body>'
```

parse_percent_directives()

Parses this compute cell's percent directives, determines its system (if any), and returns the "cleaned" input text.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '%hide\n%maxima\n%pi+3', '5', None)
sage: C.parse_percent_directives()
u'%pi+3'
sage: C.percent_directives()
[u'hide', u'maxima']
```

percent_directives()

Returns a list of this compute cell's percent directives.

OUTPUT:

- a list of strings

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '%hide\n%maxima\n2+3', '5', None)
sage: C.percent_directives()
[u'hide', u'maxima']
```

plain_text(*ncols=0*, *prompts=True*, *max_out=None*)

Returns the plain text version of this compute cell.

INPUT:

- `ncols` - an integer (default: 0); the number of word wrap columns

- `prompts` - a boolean (default: False); whether to strip interpreter prompts from the beginning of each line
- `max_out` - an integer (default: None); the maximum number of characters to return

OUTPUT:

- `plaintext_output` - Plaintext string of the cell

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: len(C.plain_text())
11
```

process_cell_urls(*urls*)

Processes this compute cell's `'cell://.*?'` URLs, replacing the protocol with the cell's path and appending the version number to prevent cached copies from shadowing the updated copy.

INPUT:

- `urls` - a string; the URLs to process

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.process_cell_urls('"cell://foobar"')
'/home/sage/0/cells/0/foobar?...'
```

sage()

Returns the `sage` instance for this compute cell(?).

OUTPUT:

- an instance of `sage`

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.sage() is None
True
```

set_asap(*asap*)

Sets whether to evaluate this compute cell as soon as possible (ASAP).

INPUT:

- `asap` - a boolean convertible

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.is_asap()
False
sage: C.set_asap(True)
sage: C.is_asap()
True
```

set_cell_output_type(*typ='wrap'*)
Sets this compute cell's output type.

INPUT:

- `typ` - a string (default: 'wrap'); the target output type

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.cell_output_type()
'wrap'
sage: C.set_cell_output_type('nowrap')
sage: C.cell_output_type()
'nowrap'
```

set_changed_input_text(*new_text*)
Updates this compute cell's changed input text. Note: This does not update the version of the cell. It's typically used, e.g., for tab completion.

INPUT:

- `new_text` - a string; the new changed input text

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.set_changed_input_text('3+3')
sage: C.input_text()
u'3+3'
sage: C.changed_input_text()
u'3+3'
```

set_input_text(*input*)
Sets the input text of this compute cell.

INPUT:

- `input` - a string; the new input text

TODO: Add doctests for the code dealing with interact.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = W.new_cell_after(0, "2^2")
sage: C.evaluate()
sage: W.check_comp(wait=9999)      # random output -- depends on computer speed
('d', Cell 1: in=2^2, out=
4
)
sage: initial_version=C.version()
sage: C.set_input_text('3+3')
sage: C.input_text()
u'3+3'
sage: C.evaluated()
False
sage: C.version()-initial_version
1
sage: W.quit()
sage: nb.delete()
```

set_introspect (*before_prompt*, *after_prompt*)
Set this compute cell's introspection text.

INPUT:

- before_prompt - a string
- after_prompt - a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.set_introspect("a", "b")
sage: C.introspect()
['a', 'b']
```

set_introspect_html (*html*, *completing=False*, *raw=False*)
Sets this compute cell's introspection text.

INPUT:

- html - a string; the updated text
- completing - a boolean (default: False); whether the completions menu is open
- raw - a boolean (default: False)

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, 'sage?', '', W)
sage: C.introspect()
False
sage: C.evaluate(username='sage')
sage: W.check_comp(9999)      # random output -- depends on computer speed
('d', Cell 0: in=sage?, out=)
sage: C.set_introspect_html('foobar')
sage: C.introspect_html()
u'foobar'
sage: C.set_introspect_html('`foobar`')
sage: C.introspect_html()
u'`foobar`'
sage: C.set_introspect_html('ěščřžýáíéd')
sage: C.introspect_html()
u'ěščřž\xfd\xe1\xed\xe9d'
sage: W.quit()
sage: nb.delete()
```

set_no_output (*no_output*)
Sets whether this is a "no output" compute cell, i.e., we don't care about its output.

INPUT:

- no_output - a boolean convertible

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.is_no_output()
False
sage: C.set_no_output(True)
```

```
sage: C.is_no_output()
True
```

set_output_text(*output*, *html*, *sage=None*)
Sets this compute cell's output text.

INPUT:

- `output` - a string; the updated output text
- `html` - a string; updated output HTML
- `sage` - a `sage` instance (default: None); the sage instance to use for this cell(?)

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: len(C.plain_text())
11
sage: C.set_output_text('10', '10')
sage: len(C.plain_text())
12
```

stop_interacting()
Stops `sagenb.notebook.interact.interact()` ion for this compute cell.

TODO: Add doctests.

system()
Returns the system used to evaluate this compute cell. The system is specified by a percent directive like '%maxima' at the top of a cell.

Returns None, if no system is explicitly specified. In this case, the notebook evaluates the cell using the worksheet's default system.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '%maxima\n2+3', '5', None)
sage: C.system()
u'maxima'
sage: prefixes = ['%hide', '%time', '']
sage: cells = [sagenb.notebook.cell.Cell(0, '%s\n2+3'%prefix, '5', None) for prefix in prefi
sage: [(C, C.system()) for C in cells if C.system() is not None]
[]
```

time()
Returns whether to print timing information about the evaluation of this compute cell.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.time()
False
sage: C = sagenb.notebook.cell.Cell(0, '%time\n2+3', '5', None)
sage: C.time()
True
```

unset_introspect()
Clears this compute cell's introspection text.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, 'sage?', '', W)
sage: C.introspect()
False
sage: C.evaluate(username='sage')
sage: W.check_comp(9999)      # random output -- depends on computer speed
('d', Cell 0: in=sage?, out=)
sage: C.introspect()
[u'sage?', '']
sage: C.unset_introspect()
sage: C.introspect()
False
sage: W.quit()
sage: nb.delete()
```

update_html_output(*output=''*)
Updates this compute cell's the file list with HTML-style links or embeddings.

For interactive cells, the HTML output section is always empty, mainly because there is no good way to distinguish content (e.g., images in the current directory) that goes into the interactive template and content that would go here.

INPUT:

- `output` - a string (default: ''); the new output

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, 'plot(sin(x),0,5)', '', W)
sage: C.evaluate()
sage: W.check_comp(wait=9999)      # random output -- depends on computer speed
('d', Cell 0: in=plot(sin(x),0,5), out=
<html><font color='black'><img src='cell://sage0.png'></font></html>

)
sage: C.update_html_output()
sage: C.output_html()      # random output -- depends on computer speed
'<img src="/home/sage/0/cells/0/sage0.png?..."></font></html>'
sage: W.quit()
sage: nb.delete()
```

url_to_self()
Returns a notebook URL for this compute cell.

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
```

```
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.url_to_self()
'/home/sage/0/cells/0'
```

version()

Returns this compute cell's version number.

OUTPUT:

- an integer

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: initial_version=C.version() #random
sage: C.set_input_text('2+3')
sage: C.version()-initial_version
1
```

word_wrap_cols()

Returns the number of columns for word wrapping this compute cell. This defaults to 70, but the default setting for a notebook is 72.

OUTPUT:

- an integer

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', None)
sage: C.word_wrap_cols()
70
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.word_wrap_cols()
72
sage: nb.delete()
```

class sagenb.notebook.cell.**Cell_generic**(*id*, *worksheet*)

Bases: object

Creates a new generic cell.

INPUT:

- `id` - an integer or string; this cell's ID
- `worksheet` - a `sagenb.notebook.worksheet.Worksheet` instance; this cell's parent worksheet

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell_generic(0, None)
sage: isinstance(C, sagenb.notebook.cell.Cell_generic)
True
sage: isinstance(C, sagenb.notebook.cell.TextCell)
False
sage: isinstance(C, sagenb.notebook.cell.Cell)
False
```

id()
Returns this generic cell's ID.

OUTPUT:

•an integer or string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell_generic(0, None)
sage: C.id()
0
sage: C = sagenb.notebook.cell.Cell('blue', '2+3', '5', None)
sage: C.id()
'blue'
sage: C = sagenb.notebook.cell.TextCell('yellow', '2+3', None)
sage: C.id()
'yellow'
```

is_auto_cell()
Returns whether this is an automatically evaluated generic cell. This is always false for Cell_generics and TextCells.

OUTPUT:

•a boolean

EXAMPLES:

```
sage: G = sagenb.notebook.cell.Cell_generic(0, None)
sage: T = sagenb.notebook.cell.TextCell(0, 'hello!', None)
sage: [X.is_auto_cell() for X in (G, T)]
[False, False]
```

is_compute_cell()
Returns whether this generic cell is a compute cell, i.e., an instance of Cell.

OUTPUT:

•a boolean

EXAMPLES:

```
sage: G = sagenb.notebook.cell.Cell_generic(0, None)
sage: T = sagenb.notebook.cell.TextCell(0, 'hello!', None)
sage: C = sagenb.notebook.cell.Cell(0, '2+4', '6', None)
sage: [X.is_compute_cell() for X in (G, T, C)]
[False, False, True]
```

is_interactive_cell()
Returns whether this generic cell uses sagenb.notebook.interact.interact() as a function call or decorator.

OUTPUT:

•a boolean

EXAMPLES:

```
sage: G = sagenb.notebook.cell.Cell_generic(0, None)
sage: T = sagenb.notebook.cell.TextCell(0, 'hello!', None)
sage: [X.is_auto_cell() for X in (G, T)]
[False, False]
```

is_last()
Returns whether this generic cell is the last cell in its worksheet object.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = W.new_cell_after(0, "2^2"); C
Cell 2: in=2^2, out=
sage: C.is_last()
True
sage: C = W.get_cell_with_id(0)
sage: C.is_last()
False
sage: nb.delete()
```

is_text_cell()
Returns whether this generic cell is a text cell, i.e., an instance of TextCell.

OUTPUT:

- a boolean

EXAMPLES:

```
sage: G = sagenb.notebook.cell.Cell_generic(0, None)
sage: T = sagenb.notebook.cell.TextCell(0, 'hello!', None)
sage: C = sagenb.notebook.cell.Cell(0, '2+4', '6', None)
sage: [X.is_text_cell() for X in (G, T, C)]
[False, True, False]
```

next_id()
Returns the ID of the next cell in this generic cell's worksheet object. If this cell is *not* in the worksheet, it returns the ID of the worksheet's *first* cell. If this *is* the last cell, it returns its *own* ID.

OUTPUT:

- an integer or string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = W.new_cell_after(1, "2^2")
sage: C = W.get_cell_with_id(1)
sage: C.next_id()
2
sage: C = W.get_cell_with_id(2)
sage: C.next_id()
2
sage: nb.delete()
```

notebook()
Returns this generic cell's associated notebook object.

OUTPUT:

- a sagenb.notebook.notebook.Notebook instance

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.notebook() is nb
True
sage: nb.delete()
```

proxied_id()

Returns the ID of the cell for which this generic cell is a proxy. If this cell does not have such an ID, it returns the cell's own ID.

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell_generic('self_stand_in', None)
sage: [C.id(), C.proxied_id()]
['self_stand_in', 'self_stand_in']
```

set_id(*id*)

Sets this generic cell's ID.

INPUT:

- id - an integer or string; the new ID

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell_generic(0, None)
sage: C.id()
0
sage: C.set_id('phone')
sage: C.id()
'phone'
```

set_proxied_id(*proxied_id*)

Sets, for this generic cell, the ID of the cell that it proxies.

INPUT:

- proxied_id - an integer or string; the proxied cell's ID

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell_generic('understudy', None)
sage: [C.id(), C.proxied_id()]
['understudy', 'understudy']
sage: C.set_proxied_id('principal')
sage: [C.id(), C.proxied_id()]
['understudy', 'principal']
```

set_worksheet(*worksheet*, *id=None*)

Sets this generic cell's worksheet object and, optionally, its ID.

INPUT:

- worksheet - a sagenb.notebook.worksheet.Worksheet instance; the cell's new worksheet object
- id - an integer or string (default: None); the cell's new ID

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell_generic(0, None)
sage: W = "worksheet object"
sage: C.set_worksheet(W)
sage: C.worksheet()
'worksheet object'
```

worksheet()

Returns this generic cell's worksheet object.

OUTPUT:

- a sagenb.notebook.worksheet.Worksheet instance

EXAMPLES:

```
sage: C = sagenb.notebook.cell.Cell_generic(0, 'worksheet object')
sage: C.worksheet()
'worksheet object'
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.worksheet() is W
True
sage: nb.delete()
```

worksheet_filename()

Returns the filename of this generic cell's worksheet object.

- publish - a boolean (default: False); whether to render a published cell

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.Cell(0, '2+3', '5', W)
sage: C.worksheet_filename()
'sage/0'
sage: nb.delete()
```

sagenb.notebook.cell.**ComputeCell**

alias of Cell

class sagenb.notebook.cell.**TextCell**(*id*, *text*, *worksheet*)

Bases: sagenb.notebook.cell.Cell_generic

Creates a new text cell.

INPUT:

- id - an integer or string; this cell's ID
- text - a string; this cell's contents
- worksheet - a sagenb.notebook.worksheet.Worksheet instance; this cell's parent worksheet

EXAMPLES:

```
sage: C = sagenb.notebook.cell.TextCell(0, '2+3', None)
sage: C == loads(dumps(C))
True
```

delete_output()

Delete all output in this text cell. This does nothing since text cells have no output.

EXAMPLES:

```
sage: C = sagenb.notebook.cell.TextCell(0, '2+3', None)
sage: C
TextCell 0: 2+3
sage: C.delete_output()
sage: C
TextCell 0: 2+3
```

edit_text()

Returns the text to be displayed for this text cell in the Edit window.

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.TextCell(0, '2+3', None)
sage: C.edit_text()
u'2+3'
```

html(*wrap=None*, *div_wrap=True*, *do_print=False*, *editing=False*, *publish=False*)

Returns HTML code for this text cell, including its contents and associated script elements.

INPUT:

- `wrap` – an integer (default: None); number of columns to wrap at (not used)
- `div_wrap` – a boolean (default: True); whether to wrap in a div (not used)
- `do_print` - a boolean (default: False); whether to render the cell for printing
- `editing` - a boolean (default: False); whether to open an editor for this cell

OUTPUT:

- a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: C = sagenb.notebook.cell.TextCell(0, '2+3', W)
sage: C.html()
u'...text_cell...2+3...'
sage: C.set_input_text("$2+3$")
```

plain_text(*prompts=False*)

Returns a plain text version of this text cell.

INPUT:

- `prompts` - a boolean (default: False); whether to strip interpreter prompts from the beginning of each line

OUTPUT:

- a string

EXAMPLES:

```
sage: C = sagenb.notebook.cell.TextCell(0, '2+3', None)
sage: C.plain_text()
u'2+3'
sage: C = sagenb.notebook.cell.TextCell(0, 'ěščřžýáíéd'Ď', None)
sage: C.plain_text()
u'ěščřž\xfd\xe1\xed\xe9d'Ď'
```

set_cell_output_type(*typ='wrap'*)

Sets this text cell's output type. This does nothing for TextCells.

INPUT:

- typ - a string (default: 'wrap'); the target output type

EXAMPLES:

```
sage: C = sagenb.notebook.cell.TextCell(0, '2+3', None)
sage: C.set_cell_output_type("wrap")
```

set_input_text(*input_text*)

Sets the input text of this text cell.

INPUT:

- input_text - a string; the new input text for this cell

EXAMPLES:

```
sage: C = sagenb.notebook.cell.TextCell(0, '2+3', None)
sage: C
TextCell 0: 2+3
sage: C.set_input_text("3+2")
sage: C
TextCell 0: 3+2
```

sagenb.notebook.cell.**format_exception**(*s0*, *ncols*)

Formats exceptions so they do not appear expanded by default.

INPUT:

- s0 - a string
- ncols - an integer; number of word wrap columns

OUTPUT:

- a string

If s0 contains "notracebacks," this function simply returns s0.

EXAMPLES:

```
sage: sagenb.notebook.cell.format_exception(sagenb.notebook.cell.TRACEBACK,80)
'\nTraceback (click to the left of this block for traceback)\n...\nTraceback (most recent call l
sage: sagenb.notebook.cell.format_exception(sagenb.notebook.cell.TRACEBACK + "notracebacks",80)
'Traceback (most recent call last):notracebacks'
```

sagenb.notebook.cell.**number_of_rows**(*txt*, *ncols*)

Returns the number of rows needed to display a string, given a maximum number of columns per row.

INPUT:

- `txt` - a string; the text to "wrap"
- `ncols` - an integer; the number of word wrap columns

OUTPUT:

- an integer

EXAMPLES:

```
sage: from sagenb.notebook.cell import number_of_rows
sage: s = "asdfasdf\nasdfasdf\n"
sage: number_of_rows(s, 8)
2
sage: number_of_rows(s, 5)
4
sage: number_of_rows(s, 4)
4
```

CHAPTER

FIVE

A WORKSHEET

A worksheet is embedded in a web page that is served by the Sage server. It is a linearly-ordered collections of numbered cells, where a cell is a single input/output block.

The worksheet module is responsible for running calculations in a worksheet, spawning Sage processes that do all of the actual work and are controlled via pexpect, and reporting on results of calculations. The state of the cells in a worksheet is stored on the file system (not in the notebook pickle sobj).

AUTHORS:

- William Stein

class sagenb.notebook.worksheet.**Worksheet**(*name=None*, *id_number=None*, *notebook_worksheet_directory=None*, *system=None*, *owner=None*, *pretty_print=False*, *auto_publish=False*, *create_directories=True*)

Bases: object

Create and initialize a new worksheet.

INPUT:

- name - string; the name of this worksheet
- **id_number - Integer; name of the directory in which the** worksheet's data is stored
- notebook_worksheet_directory - string; the directory in which the notebook object that contains this worksheet stores worksheets, i.e., nb.worksheet_directory().
- system - string; 'sage', 'gp', 'singular', etc. - the math software system in which all code is evaluated by default
- owner - string; username of the owner of this worksheet
- pretty_print - bool (default: False); whether all output is pretty printed by default.
- create_directories – bool (default: True): if True, creates various files and directories where data will be stored. This option is here only for the migrate_old_notebook method in notebook.py

EXAMPLES: We test the constructor via an indirect doctest:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: import sagenb.notebook.misc
sage: sagenb.notebook.misc.notebook = nb
sage: W = nb.create_new_worksheet('Test with unicode ščřžýáíéd', 'admin')
sage: W
admin/0: [Cell 1: in=, out=]
```

add_collaborator(*user*)
Add the given user as a collaborator on this worksheet.

INPUT:

- `user` - a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: nb.user_manager().add_user('diophantus','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Collaborator test', 'admin')
sage: W.collaborators()
[]
sage: W.add_collaborator('diophantus')
sage: W.collaborators()
['diophantus']
```

add_viewer(*user*)
Add the given user as an allowed viewer of this worksheet.

INPUT:

- `user` - string (username)

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: nb.user_manager().add_user('diophantus','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Viewer test', 'admin')
sage: W.add_viewer('diophantus')
sage: W.viewers()
['diophantus']
```

append(*L*)
x.__init__(...) initializes x; see help(type(x)) for signature

append_new_cell()
Creates and appends a new compute cell to this worksheet's list of cells.

OUTPUT:

- a new `sagenb.notebook.cell.Cell` instance

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test Edit Save', 'admin')
sage: W
admin/0: [Cell 1: in=, out=]
sage: W.append_new_cell()
Cell 2: in=, out=
sage: W
admin/0: [Cell 1: in=, out=, Cell 2: in=, out=]
```

attach(*filename*)
x.__init__(...) initializes x; see help(type(x)) for signature

attached_data_files()
Return a list of the file names of files in the worksheet data directory.

OUTPUT: list of strings

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.attached_data_files()
[]
sage: open('%s/foo.data'%W.data_directory(),'w').close()
sage: W.attached_data_files()
['foo.data']
```

attached_files()
x.__init__(...) initializes x; see help(type(x)) for signature

attached_html(*username=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

autosave(*username*)
x.__init__(...) initializes x; see help(type(x)) for signature

basic()
Output a dictionary of basic Python objects that defines the configuration of this worksheet, except the actual cells and the data files in the DATA directory and images and other data in the individual cell directories.

EXAMPLES:

```
sage: import sagenb.notebook.worksheet
sage: W = sagenb.notebook.worksheet.Worksheet('test', 0, tmp_dir(), owner='sage')
sage: sorted((W.basic().items()))
[('auto_publish', False), ('collaborators', []), ('id_number', 0), ('last_change', ('sage',
```

best_completion(*s*, *word*)
x.__init__(...) initializes x; see help(type(x)) for signature

body()
OUTPUT:

– string – Plain text representation of the body of the worksheet.

body_is_loaded()
Return True if the body if this worksheet has been loaded from disk.

cell_directory(*C*)
x.__init__(...) initializes x; see help(type(x)) for signature

cell_id_list()
Returns a list of ID's of all cells in this worksheet.

OUTPUT:

•a new list of integers and/or strings

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test Edit Save', 'admin')
```

Now we set the worksheet to have two cells with the default id of 0 and another with id 10.

```
sage: W.edit_save('{{{\n2+3\n///\n5\n}}}\n{{{id=10|\n2+8\n///\n10\n}}}')
sage: W.cell_id_list()
[0, 10]
```

cell_list()
Returns a reference to the list of this worksheet's cells.

OUTPUT:

- a list of sagenb.notebook.cell.Cell_generic instances

Note: This function loads the cell list from disk (the file worksheet.html) if it isn't available in memory.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test Edit Save', 'admin')
sage: W.edit_save('{{{\n2+3\n///\n5\n}}}\n{{{\n2+8\n///\n10\n}}}')
sage: v = W.cell_list(); v
[Cell 0: in=2+3, out=
5, Cell 1: in=2+8, out=
10]
sage: v[0]
Cell 0: in=2+3, out=
5
```

cells_directory()
Return the directory in which the cells of this worksheet are evaluated.

OUTPUT: string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.cells_directory()
'.../home/admin/0/cells'
```

check_cell(*id*)
Checks the status of a given compute cell.

INPUT:

- id - an integer or a string; the cell's ID.

OUTPUT:

- a (string, sagenb.notebook.cell.Cell)-tuple; the cell's status ('d' for "done" or 'w' for "working") and the cell itself.

check_comp(*wait=0.2*)
Check on currently computing cells in the queue.

INPUT:

- wait - float (default: 0.2); how long to wait for output.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save('{{{\n3^20\n}}}')
sage: W.cell_list()[0].evaluate()
sage: W.check_comp()      # random output -- depends on computer speed
('d', Cell 0: in=3^20, out=
3486784401
)
sage: W.quit()
sage: nb.delete()
```

check_for_system_switching(*input*, *cell*)

Check for input cells that start with `%foo`, where `foo` is an object with an eval method.

INPUT:

- `s` - a string of the code from the cell to be executed
- `C` - the cell object

EXAMPLES: First, we set up a new notebook and worksheet.

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
```

We first test running a native command in 'sage' mode and then a GAP cell within Sage mode.

```
sage: W.edit_save('{{{\n2+3\n}}}\n\n{{{\n%gap\nSymmetricGroup(5)\n}}}')
sage: c0, c1 = W.cell_list()
sage: W.check_for_system_switching(c0.cleaned_input_text(), c0)
(False, u'2+3')
sage: W.check_for_system_switching(c1.cleaned_input_text(), c1)
(True, u"print _support_.syseval(gap, u'SymmetricGroup(5)', __SAGE_TMP_DIR__)")

sage: c0.evaluate()
sage: W.check_comp()  #random output -- depends on the computer's speed
('d', Cell 0: in=2+3, out=
5
)
sage: c1.evaluate()
sage: W.check_comp()  #random output -- depends on the computer's speed
('d', Cell 1: in=%gap
SymmetricGroup(5), out=
Sym( [ 1 .. 5 ] )
)
```

Next, we run the same commands but from 'gap' mode.

```
sage: W.edit_save('{{{\n%sage\n2+3\n}}}\n\n{{{\nSymmetricGroup(5)\n}}}')
sage: W.set_system('gap')
sage: c0, c1 = W.cell_list()
sage: W.check_for_system_switching(c0.cleaned_input_text(), c0)
(False, u'2+3')
sage: W.check_for_system_switching(c1.cleaned_input_text(), c1)
(True, u"print _support_.syseval(gap, u'SymmetricGroup(5)', __SAGE_TMP_DIR__)")
sage: c0.evaluate()
sage: W.check_comp()  #random output -- depends on the computer's speed
('d', Cell 0: in=%sage
2+3, out=
```

```
5
)
sage: c1.evaluate()
sage: W.check_comp()  #random output -- depends on the computer's speed
('d', Cell 1: in=SymmetricGroup(5), out=
Sym( [ 1 .. 5 ] )
)
sage: W.quit()
sage: nb.delete()
```

clear()
x.__init__(...) initializes x; see help(type(x)) for signature

clear_queue()
x.__init__(...) initializes x; see help(type(x)) for signature

collaborator_names(*max=None*)
Returns a string of the non-owner collaborators on this worksheet.

INPUT:

- `max` - an integer. If this is specified, then only max number of collaborators are shown.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('test1', 'admin')
sage: C = W.collaborators(); C
[]
sage: C.append('sage')
sage: C.append('wstein')
sage: W.collaborator_names()
'sage, wstein'
sage: W.collaborator_names(max=1)
'sage, ...'
```

collaborators()
Return a (reference to the) list of the collaborators who can also view and modify this worksheet.

OUTPUT: list

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('test1', 'admin')
sage: C = W.collaborators(); C
[]
sage: C.append('sage')
sage: W.collaborators()
['sage']
```

completions_html(*id*, *s*, *cols=3*)
x.__init__(...) initializes x; see help(type(x)) for signature

compute_cell_id_list()
Returns a list of ID's of all compute cells in this worksheet.

OUTPUT:

- a new list of integers and/or strings

compute_cell_list()
Returns a list of this worksheet's compute cells.

OUTPUT:

- a list of sagenb.notebook.cell.Cell instances

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: W.edit_save('foo\n{{{\n2+3\n///\n5\n}}}bar\n{{{\n2+8\n///\n10\n}}}')
sage: v = W.compute_cell_list(); v
[Cell 1: in=2+3, out=
5, Cell 3: in=2+8, out=
10]
sage: v[0]
Cell 1: in=2+3, out=
5
```

compute_process_has_been_started()
Return True precisely if the compute process has been started, irregardless of whether or not it is currently churning away on a computation.

computing()
Return whether or not a cell is currently being run in the worksheet Sage process.

create_directories()
x.__init__(...) initializes x; see help(type(x)) for signature

cython_import(*cmd*, *cell*)
x.__init__(...) initializes x; see help(type(x)) for signature

data_directory()
Return path to directory where worksheet data is stored.

OUTPUT: string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.data_directory()
'.../home/admin/0/data'
```

date_edited()
Returns the date the worksheet was last edited.

delete_all_output(*username*)
Delete all the output, files included, in all the worksheet cells.

INPUT:

- username - name of the user requesting the deletion.

EXAMPLES: We create a new notebook, user, and a worksheet:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save("{{{\n2+3\n///\n5\n}}}\n{{{\nopen('afile', 'w').write('some text')\nprint
```

We have two cells:

```
sage: W.cell_list()
[Cell 0: in=2+3, out=
5, Cell 1: in=open('afile', 'w').write('some text')
print 'hello', out=
]
sage: C0 = W.cell_list()[1]
sage: open(os.path.join(C0.directory(), 'xyz'), 'w').write('bye')
sage: C0.files()
['xyz']
sage: C1 = W.cell_list()[1]
sage: C1.evaluate()
sage: W.check_comp()      # random output -- depends on computer speed
('w', Cell 1: in=open('afile', 'w').write('some text')
print 'hello', out=)
sage: W.check_comp()      # random output -- depends on computer speed
('d', Cell 1: in=open('afile', 'w').write('some text')
print 'hello', out=
hello
)
sage: W.check_comp()      # random output -- depends on computer speed
('e', None)
sage: C1.files()          # random output -- depends on computer speed
['afile']
```

We now delete the output, observe that it is gone:

```
sage: W.delete_all_output('sage')
sage: W.cell_list()
[Cell 0: in=2+3, out=, Cell 1: in=open('afile', 'w').write('some text')
print 'hello', out=]
sage: C0.files(), C1.files()
([], [])
```

If an invalid user tries to delete all output, a ValueError is raised:

```
sage: W.delete_all_output('hacker')
Traceback (most recent call last):
...
ValueError: user 'hacker' not allowed to edit this worksheet
```

Clean up:

```
sage: W.quit()
sage: nb.delete()
```

delete_cell_with_id(*id*)

Deletes a cell from this worksheet's cell list. This also deletes the cell's output and files.

INPUT:

- `id` - an integer or string; the cell's ID

OUTPUT:

- an integer or string; ID of the preceding cell

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test Delete Cell', 'admin')
```

```
sage: W.edit_save('{{{id=foo|\n2+3\n///\n5\n}}}\n{{{id=9|\n2+8\n///\n10\n}}}{{{id=dont_delet
sage: W.cell_id_list()
['foo', 9, 'dont_delete_me']
sage: C = W.cell_list()[1]           # save a reference to the cell
sage: C.output_text(raw=True)
u'\n10'
sage: open(os.path.join(C.directory(), 'bar'), 'w').write('hello')
sage: C.files()
['bar']
sage: C.files_html('')
u'<a target="_new" href=".../cells/9/bar" class="file_link">bar</a>'
sage: W.delete_cell_with_id(C.id())
'foo'
sage: C.output_text(raw=True)
u''
sage: C.files()
[]
sage: W.cell_id_list()
['foo', 'dont_delete_me']
```

delete_cells_directory()

Delete the directory in which all the cell computations occur.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save('{{{\n3^20\n}}}')
sage: W.cell_list()[0].evaluate()
sage: W.check_comp()     # random output -- depends on computer speed
sage: sorted(os.listdir(W.directory()))
['cells', 'data', 'worksheet.html', 'worksheet_conf.pickle']
sage: W.save_snapshot('admin')
sage: sorted(os.listdir(W.directory()))
['cells', 'data', 'snapshots', 'worksheet.html', 'worksheet_conf.pickle']
sage: W.delete_cells_directory()
sage: sorted(os.listdir(W.directory()))
['data', 'snapshots', 'worksheet.html', 'worksheet_conf.pickle']
sage: W.quit()
sage: nb.delete()
```

delete_notebook_specific_data()

Delete data from this worksheet this is specific to a certain notebook. This means deleting the attached files, collaborators, and viewers.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: nb.user_manager().add_user('hilbert','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('test1', 'admin')
sage: W.add_viewer('hilbert')
sage: W.delete_notebook_specific_data()
sage: W.viewers()
[]
sage: W.add_collaborator('hilbert')
sage: W.collaborators()
```

```
['admin', 'hilbert']
sage: W.delete_notebook_specific_data()
sage: W.collaborators()
['admin']
```

delete_user(*user*)

Delete a user from having any view or ownership of this worksheet.

INPUT:

- `user` - string; the name of a user

EXAMPLES: We create a notebook with 2 users and 1 worksheet that both view.

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('wstein','sage','wstein@sagemath.org',force=True)
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.new_worksheet_with_title_from_text('Sage', owner='sage')
sage: W.add_viewer('wstein')
sage: W.owner()
'sage'
sage: W.viewers()
['wstein']
```

We delete the sage user from the worksheet W. This makes wstein the new owner.

```
sage: W.delete_user('sage')
sage: W.viewers()
['wstein']
sage: W.owner()
'wstein'
```

Then we delete wstein from W, which makes the owner None.

```
sage: W.delete_user('wstein')
sage: W.owner() is None
True
sage: W.viewers()
[]
```

Finally, we clean up.

```
sage: nb.delete()
```

detach(*filename*)

x.__init__(...) initializes x; see help(type(x)) for signature

directory()

Return the full path to the directory where this worksheet is stored.

OUTPUT: string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.directory()
'.../home/admin/0'
```

docbrowser()

Return True if this is a docbrowser worksheet.

OUTPUT: bool

EXAMPLES: We first create a standard worksheet for which docbrowser is of course False:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('test1', 'admin')
sage: W.docbrowser()
False
```

We create a worksheet for which docbrowser is True:

```
sage: W = nb.create_new_worksheet('doc_browser_0', '_sage_')
sage: W.docbrowser()
True
```

download_name()

Return the download name of this worksheet.

edit_save(*text*, *ignore_ids=False*)

Set the contents of this worksheet to the worksheet defined by the plain text string text, which should be a sequence of HTML and code blocks.

INPUT:

- `text` - a string
- `ignore_ids` - bool (default: False); if True ignore all the IDs in the {{{}}} code block.

EXAMPLES:

We create a new test notebook and a worksheet.

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test Edit Save', 'sage')
```

We set the contents of the worksheet using the edit_save command.

```
sage: W.edit_save('{{{\n2+3\n///\n5\n}}}\n{{{\n2+8\n///\n10\n}}}')
sage: W
sage/0: [Cell 0: in=2+3, out=
5, Cell 1: in=2+8, out=
10]
sage: W.name()
u'Test Edit Save'
```

We check that loading a worksheet whose last cell is a `TextCell` properly increments the worksheet's cell count (see Sage trac ticket #8443).

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage', 'sage', 'sage@sagemath.org', force=True)
sage: W = nb.create_new_worksheet('Test trac #8443', 'sage')
sage: W.edit_save('{{{\n1+1\n///\n}}}')
sage: W.cell_id_list()
[0]
sage: W.next_id()
1
sage: W.edit_save("{{{\n1+1\n///\n}}}\n\n<p>a text cell</p>")
sage: len(set(W.cell_id_list())) == 3
True
sage: W.cell_id_list()
[0, 1, 2]
```

```
sage: W.next_id()
3
```

edit_save_old_format (*text*, *username=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

edit_text ()
Returns a plain-text version of the worksheet with {{{}}} wiki-formatting, suitable for hand editing.

enqueue (*C*, *username=None*, *next=False*)
Queue a cell for evaluation in this worksheet.

INPUT:

- C - a sagenb.notebook.cell.Cell instance
- **username - a string (default: None); the name of the** user evaluating this cell (mainly used for login)
- next - a boolean (default: False); ignored

Note: If C.is_asap() is True, then we put C as close to the beginning of the queue as possible, but after all "asap" cells. Otherwise, C goes at the end of the queue.

---

eval_asap_no_output (*cmd*, *username=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

everyone_has_deleted_this_worksheet ()
Return True if all users have deleted this worksheet, so we know we can safely purge it from disk.

OUTPUT: bool

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.everyone_has_deleted_this_worksheet()
False
sage: W.move_to_trash('admin')
sage: W.everyone_has_deleted_this_worksheet()
True
```

filename ()
Return the filename (really directory) where the files associated to this worksheet are stored.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.filename()
'admin/0'
sage: os.path.isdir(os.path.join(nb._dir, 'home', W.filename()))
True
```

filename_without_owner ()
Return the part of the worksheet filename after the last /, i.e., without any information about the owner of this worksheet.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.filename_without_owner()
'0'
sage: W.filename()
'admin/0'
```

get_cell_system(*cell*)
Returns the system that will run the input in cell. This defaults to worksheet's system if there is not one specifically given in the cell.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save('{{{\n2+3\n}}}\n\n{{{\n%gap\nSymmetricGroup(5)\n}}}')
sage: c0, c1 = W.cell_list()
sage: W.get_cell_system(c0)
'sage'
sage: W.get_cell_system(c1)
u'gap'
sage: W.edit_save('{{{\n%sage\n2+3\n}}}\n\n{{{\nSymmetricGroup(5)\n}}}')
sage: W.set_system('gap')
sage: c0, c1 = W.cell_list()
sage: W.get_cell_system(c0)
u'sage'
sage: W.get_cell_system(c1)
'gap'
```

get_cell_with_id(*id*)
Get a pre-existing cell with this id, or creates a new one with it.

get_cell_with_id_or_none(*id*)
Gets a pre-existing cell with this id, or returns None.

get_snapshot_text_filename(*name*)
x.__init__(...) initializes x; see help(type(x)) for signature

has_published_version()
Return True if there is a published version of this worksheet.

OUTPUT: bool

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: P = nb.publish_worksheet(W, 'admin')
sage: P.has_published_version()
False
sage: W.has_published_version()
True
```

hide_all()
x.__init__(...) initializes x; see help(type(x)) for signature

html(*do_print=False*, *publish=False*, *username=None*)
INPUT:

•publish - a boolean stating whether the worksheet is published

•do_print - a boolean

OUTPUT:

•string – the HTML for the worksheet

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: W.html()
u'...cell_input_active...worksheet_cell_list...cell_1...cell_output_html_1...'
```

html_cell_list(*do_print=False*, *username=None*)

INPUT:

•do_print - a boolean

OUTPUT:

•string – the HTML for the list of cells

html_ratings_info(*username=None*)

Return html that renders to give a summary of how this worksheet has been rated.

OUTPUT:

•`string` – a string of HTML as a bunch of table rows.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.rate(0, 'this lacks content', 'riemann')
sage: W.rate(3, 'this is great', 'hilbert')
sage: W.html_ratings_info()
u'...hilbert...3...this is great...this lacks content...'
```

html_time_last_edited(*username=None*)

x.__init__(...) initializes x; see help(type(x)) for signature

html_time_nice_edited(*username=None*)

Returns a "nice" html time since last edit.

If the last edit was in the last 24 hours, return a "x hours ago". Otherwise, return a specific date.

html_time_since_last_edited(*username=None*)

x.__init__(...) initializes x; see help(type(x)) for signature

hunt_file(*filename*)

x.__init__(...) initializes x; see help(type(x)) for signature

id_number()

Return the id number of this worksheet, which is an integer.

EXAMPLES:

```
sage: from sagenb.notebook.worksheet import Worksheet
sage: W = Worksheet('test', 2, tmp_dir(), owner='sageuser')
sage: W.id_number()
2
```

```
sage: type(W.id_number())
<type 'int'>
```

increase_state_number()
x.__init__(...) initializes x; see help(type(x)) for signature

initialize_sage()
x.__init__(...) initializes x; see help(type(x)) for signature

input_text()
Return text version of the input to the worksheet.

interrupt(*callback=None*, *timeout=1*)
Interrupt all currently queued up calculations.

INPUT:

- `timeout` – time to wait for interruption to succeed
- `callback` – callback to be called. Called with True if interrupt succeeds, else called with False.

OUTPUT:

- `deferred` - a Deferred object with the given callbacks and errbacks

EXAMPLES: We create a worksheet and start a large factorization going:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.edit_save('{{{\nfactor(2^997-1)\n}}}')
sage: W.cell_list()[0].evaluate()
```

It's running still:

```
sage: W.check_comp()
('w', Cell 0: in=factor(2^997-1), out=...)
```

We interrupt it successfully.

```
sage: W.interrupt()          # not tested -- needs running reactor
True
```

Now we check and nothing is computing.

```
sage: W.check_comp()         # random -- could fail on heavily loaded machine
('e', None)
```

Clean up.

```
sage: W.quit()
sage: nb.delete()
```

is_active(*user*)
Return True if this worksheet is active for the given user.

INPUT:

- `user` - string

OUTPUT: bool

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Active Test', 'admin')
sage: W.is_active('admin')
True
sage: W.move_to_archive('admin')
sage: W.is_active('admin')
False
```

is_archived(*user*)
Return True if this worksheet is archived for the given user.

INPUT:

- `user` - string

OUTPUT: bool

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Archived Test', 'admin')
sage: W.is_archived('admin')
False
sage: W.move_to_archive('admin')
sage: W.is_archived('admin')
True
```

is_auto_publish()
Returns True if this worksheet should be automatically published.

is_collaborator(*user*)
x.__init__(...) initializes x; see help(type(x)) for signature

is_last_id_and_previous_is_nonempty(*id*)
x.__init__(...) initializes x; see help(type(x)) for signature

is_only_viewer(*user*)
x.__init__(...) initializes x; see help(type(x)) for signature

is_owner(*username*)
x.__init__(...) initializes x; see help(type(x)) for signature

is_published()
Return True if this worksheet is a published worksheet.

OUTPUT:

- `bool` - whether or not owner is 'pub'

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.is_published()
False
sage: W.set_owner('pub')
sage: W.is_published()
True
```

is_publisher(*username*)
Return True if username is the username of the publisher of this worksheet, assuming this worksheet was published.

INPUT:

- `username` - string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: P = nb.publish_worksheet(W, 'admin')
sage: P.is_publisher('hearst')
False
sage: P.is_publisher('admin')
True
```

is_rater(*username*)
Return True is the user with given username has rated this worksheet.

INPUT:

- `username` - string

OUTPUT: bool

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.rate(0, 'this lacks content', 'riemann')
sage: W.is_rater('admin')
False
sage: W.is_rater('riemann')
True
```

is_trashed(*user*)
Return True if this worksheet is in the trash for the given user.

INPUT:

- `user` - string

OUTPUT: bool

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Trash Test', 'admin')
sage: W.is_trashed('admin')
False
sage: W.move_to_trash('admin')
sage: W.is_trashed('admin')
True
```

is_viewer(*user*)
x.__init__(...) initializes x; see help(type(x)) for signature

last_change()
Return information about when this worksheet was last changed and by whom.

OUTPUT:

- `username` – string of username who last edited this worksheet
- `float` – seconds since epoch of the time when this worksheet was last edited

last_compute_walltime()
x.__init__(...) initializes x; see help(type(x)) for signature

last_edited()
x.__init__(...) initializes x; see help(type(x)) for signature

last_to_edit()
x.__init__(...) initializes x; see help(type(x)) for signature

limit_snapshots()
This routine will limit the number of snapshots of a worksheet, as specified by a hard-coded value below.

Prior behavior was to allow unlimited numbers of snapshots and so this routine will not delete files created prior to this change.

This assumes snapshot names correspond to the `time.time()` method used to create base filenames in seconds in UTC time, and that there are no other extraneous files in the directory.

load_any_changed_attached_files(*s*)
Modify `s` by prepending any necessary load commands corresponding to attached files that have changed.

load_path()
x.__init__(...) initializes x; see help(type(x)) for signature

move_out_of_trash(*user*)
Exactly the same as set_active(user).

INPUT:

- `user` - string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Active Test', 'admin')
sage: W.move_to_trash('admin')
sage: W.is_active('admin')
False
sage: W.move_out_of_trash('admin')
sage: W.is_active('admin')
True
```

move_to_archive(*user*)
Move this worksheet to be archived for the given user.

INPUT:

- `user` - string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Archive Test', 'admin')
sage: W.move_to_archive('admin')
sage: W.is_archived('admin')
True
```

move_to_trash(*user*)
Move this worksheet to the trash for the given user.

INPUT:

- `user` - string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Trash Test', 'admin')
sage: W.move_to_trash('admin')
sage: W.is_trashed('admin')
True
```

name(*username=None*)
Return the name of this worksheet.

OUTPUT: string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.name()
u'A Test Worksheet'
sage: W = nb.create_new_worksheet('ěščřžýáíéd'Ď', 'admin')
sage: W.name()
u'ěščřž\xfd\xe1\xed\xe9d'Ď'
```

new_cell_after(*id*, *input=''*)
Inserts a new compute cell into this worksheet's cell list after the cell with the given ID. If the ID does not match any cell, it inserts the new cell at the end of the list.

INPUT:

- `id` - an integer or a string; the ID of the cell to find
- `input` - a string (default: ''); the new cell's input text

OUTPUT:

- a new `sagenb.notebook.cell.Cell` instance

new_cell_before(*id*, *input=''*)
Inserts a new compute cell before a cell with the given ID. If the ID does not match any cell in this worksheet's list, it inserts a new cell at the end.

INPUT:

- `id` - an integer or a string; the ID of the cell to find
- `input` - a string (default: ''); the new cell's input text

OUTPUT:

- a new `sagenb.notebook.cell.Cell` instance

new_text_cell_after(*id*, *input=''*)
Inserts a new text cell into this worksheet's cell list after the cell with the given ID. If the ID does not match any cell, it inserts the new cell at the end of the list.

INPUT:

- `id` - an integer or a string; the ID of the cell to find
- `input` - a string (default: ''); the new cell's input text

OUTPUT:

- a new `sagenb.notebook.cell.TextCell` instance

new_text_cell_before(*id*, *input=''*)

Inserts a new text cell before the cell with the given ID. If the ID does not match any cell in this worksheet's list, it inserts a new cell at the end.

INPUT:

- `id` - an integer or a string; the ID of the cell to find
- `input` - a string (default: ''); the new cell's input text

OUTPUT:

- a new `sagenb.notebook.cell.TextCell` instance

next_block_id()

x.__init__(...) initializes x; see help(type(x)) for signature

next_hidden_id()

x.__init__(...) initializes x; see help(type(x)) for signature

next_id()

x.__init__(...) initializes x; see help(type(x)) for signature

notebook()

Return the notebook that contains this worksheet.

OUTPUT: a Notebook object.

Note: This really returns the Notebook object that is set as a global variable of the misc module. This is done *even* in the Flask version of the notebook as it is set in func:$sagenb.notebook.notebook.load_notebook$.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.notebook()
<...sagenb.notebook.notebook.Notebook...>
sage: W.notebook() is sagenb.notebook.misc.notebook
True
```

onload_id_list()

Returns a list of ID's of cells the remote client should evaluate after the worksheet loads. Unlike '%auto' cells, which the server chooses to evaluate, onload cells fire only after the client sends a request. Currently, we use onload cells to set up published interacts.

OUTPUT:

- a new list of integer and/or string IDs

owner()

x.__init__(...) initializes x; see help(type(x)) for signature

ping(*username*)
x.__init__(...) initializes x; see help(type(x)) for signature

plain_text(*prompts=False*, *banner=True*)
Return a plain-text version of the worksheet.

INPUT:

- `prompts` - if True format for inclusion in docstrings.

postprocess_output(*out*, *C*)
x.__init__(...) initializes x; see help(type(x)) for signature

preparse(*s*)
Return preparsed version of input code `s`, ready to be sent to the Sage process for evaluation. The output is a "safe string" (no funny characters).

INPUT:

- `s` – a string

OUTPUT:

- a string

preparse_input(*input*, *C*)
x.__init__(...) initializes x; see help(type(x)) for signature

preparse_introspection_input(*input*, *C*, *introspect*)
x.__init__(...) initializes x; see help(type(x)) for signature

preparse_nonswitched_input(*input*)
Preparse the input to a Sage Notebook cell.

INPUT:

- `input` – a string

OUTPUT:

- a string

pretty_print()
Return True if output should be pretty printed by default.

OUTPUT:

- `bool` - True of False

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.pretty_print()
False
sage: W.set_pretty_print('true')
sage: W.pretty_print()
True
sage: W.quit()
sage: nb.delete()
```

published_version()
If this worksheet was published, return the published version of this worksheet. Otherwise, raise a ValueError.

OUTPUT: a worksheet (or raise a ValueError)

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: P = nb.publish_worksheet(W, 'admin')
sage: W.published_version() is P
True
```

publisher()

Return username of user that published this worksheet.

OUTPUT: string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: S = nb.publish_worksheet(W, 'admin')
sage: S.publisher()
'admin'
```

queue()

x.__init__(...) initializes x; see help(type(x)) for signature

queue_id_list()

x.__init__(...) initializes x; see help(type(x)) for signature

quit()

x.__init__(...) initializes x; see help(type(x)) for signature

quit_if_idle(*timeout*)

Quit the worksheet process if it has been "idle" for more than `timeout` seconds, where idle is by definition that the worksheet has not reported back that it is actually computing. I.e., an ignored worksheet process (since the user closed their browser) is also considered idle, even if code is running.

rate(*x*, *comment*, *username*)

Set the rating on this worksheet by the given user to x and also set the given comment.

INPUT:

- `x` - integer
- `comment` - string
- `username` - string

EXAMPLES: We create a worksheet and rate it, then look at the ratings.

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.rate(3, 'this is great', 'hilbert')
sage: W.ratings()
[('hilbert', 3, 'this is great')]
```

Note that only the last rating by a user counts:

```
sage: W.rate(1, 'this lacks content', 'riemann')
sage: W.rate(0, 'this lacks content', 'riemann')
```

```
sage: W.ratings()
[('hilbert', 3, 'this is great'), ('riemann', 0, 'this lacks content')]
```

rating()

Return overall average rating of self.

OUTPUT: float or the int -1 to mean "not rated"

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.rating()
-1
sage: W.rate(0, 'this lacks content', 'riemann')
sage: W.rate(3, 'this is great', 'hilbert')
sage: W.rating()
1.5
```

ratings()

Return all the ratings of this worksheet.

OUTPUT:

- `list` - a reference to the list of ratings.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.ratings()
[]
sage: W.rate(0, 'this lacks content', 'riemann')
sage: W.rate(3, 'this is great', 'hilbert')
sage: W.ratings()
[('riemann', 0, 'this lacks content'), ('hilbert', 3, 'this is great')]
```

reconstruct_from_basic(*obj*, *notebook_worksheet_directory=None*)

Reconstruct as much of the worksheet's configuration as possible from the properties that happen to be set in the basic dictionary obj.

INPUT:

- `obj` – a dictionary of basic Python objects
- `notebook_worksheet_directory` – must be given if `id_number` is a key of obj; otherwise not.

EXAMPLES:

```
sage: import sagenb.notebook.worksheet
sage: W = sagenb.notebook.worksheet.Worksheet('test', 0, tmp_dir(), system='gap', owner='sag
sage: W.new_cell_after(0)
Cell 1: in=, out=
sage: b = W.basic()
sage: W0 = sagenb.notebook.worksheet.Worksheet()
sage: W0.reconstruct_from_basic(b, tmp_dir())
sage: W0.basic() == W.basic()
True
```

record_edit(*user*)
x.__init__(...) initializes x; see help(type(x)) for signature

reset_interact_state()
Reset the interact state of this worksheet.

restart_sage()
Restart Sage kernel.

revert_to_last_saved_state()
x.__init__(...) initializes x; see help(type(x)) for signature

revert_to_snapshot(*name*)
x.__init__(...) initializes x; see help(type(x)) for signature

sage()
Return a started up copy of Sage initialized for computations.

If this is a published worksheet, just return None, since published worksheets must not have any compute functionality.

OUTPUT: a Sage interface

satisfies_search(*search*)
Return True if all words in search are in the saved text of the worksheet.

INPUT:

- `search` - a string that describes a search query, i.e., a space-separated collections of words.

OUTPUT:

- a boolean

save(*conf_only=False*)
x.__init__(...) initializes x; see help(type(x)) for signature

save_snapshot(*user*, *E=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

set_active(*user*)
Set his worksheet to be active for the given user.

INPUT:

- `user` - string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Active Test', 'admin')
sage: W.move_to_archive('admin')
sage: W.is_active('admin')
False
sage: W.set_active('admin')
sage: W.is_active('admin')
True
```

set_auto_publish(*x*)
x.__init__(...) initializes x; see help(type(x)) for signature

set_body(*body*)
x.__init__(...) initializes x; see help(type(x)) for signature

set_cell_counter()
x.__init__(...) initializes x; see help(type(x)) for signature

set_collaborators(*v*)
Set the list of collaborators to those listed in the list v of strings.

INPUT:

- v - a list of strings

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: nb.user_manager().add_user('hilbert','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('test1', 'admin')
sage: W.set_collaborators(['sage', 'admin', 'hilbert', 'sage'])
```

Note that repeats are not added multiple times and admin - the owner - isn't added:

```
sage: W.collaborators()
['hilbert', 'sage']
```

set_filename(*filename*)
Set the worksheet filename (actually directory).

INPUT:

- filename - string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.filename()
'admin/0'
sage: W.set_filename('admin/10')
sage: W.filename()
'admin/10'
```

set_filename_without_owner(*nm*)
Set this worksheet filename (actually directory) by getting the owner from the pre-stored owner via self.owner().

INPUT:

- nm - string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.filename()
'admin/0'
sage: W.set_filename_without_owner('5')
sage: W.filename()
'admin/5'
```

set_last_change(*username*, *tm*)
Set the time and who last changed this worksheet.

INPUT:

- `username` – (string) name of a user
- `tm` – (float) seconds since epoch

EXAMPLES:

```
sage: W = sagenb.notebook.worksheet.Worksheet('test', 0, tmp_dir(), owner='sage')
sage: W.last_change()
('sage', ...)
sage: W.set_last_change('john', 1255029800)
sage: W.last_change()
('john', 1255029800.0)
```

We make sure these other functions have been correctly updated:

```
sage: W.last_edited()
1255029800.0
sage: W.last_to_edit()
'john'
sage: W.date_edited() # Output depends on timezone
time.struct_time(tm_year=2009, tm_mon=10, ...)
sage: t = W.time_since_last_edited() # just test that call works
```

set_name(*name*)

Set the name of this worksheet.

INPUT:

- `name` - string

EXAMPLES: We create a worksheet and change the name:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.set_name('A renamed worksheet')
sage: W.name()
u'A renamed worksheet'
```

set_not_computing()

x.__init__(...) initializes x; see help(type(x)) for signature

set_owner(*owner*)

x.__init__(...) initializes x; see help(type(x)) for signature

set_pretty_print(*check='false'*)

Set whether or not output should be pretty printed by default.

INPUT:

- `check` - string (default: 'false'); either 'true' or 'false'.

Note: The reason the input is a string and lower case instead of a Python bool is because this gets called indirectly from JavaScript. (And, Jason Grout wrote this and didn't realize how unpythonic this design is - it should be redone to use True/False.)

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
```

```
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.set_pretty_print('false')
sage: W.pretty_print()
False
sage: W.set_pretty_print('true')
sage: W.pretty_print()
True
sage: W.quit()
sage: nb.delete()
```

set_published_version(*filename*)
Set the published version of this worksheet to be the worksheet with given filename.

INPUT:

- `filename` - string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: P = nb.publish_worksheet(W, 'admin')  # indirect test
sage: W._Worksheet__published_version
'pub/0'
sage: W.set_published_version('pub/1')
sage: W._Worksheet__published_version
'pub/1'
```

set_system(*system='sage'*)
Set the math software system in which input is evaluated by default.

INPUT:

- `system` - string (default: 'sage')

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.set_system('magma')
sage: W.system()
'magma'
```

set_tags(*tags*)
Set the tags – for now we ignore everything except ACTIVE, ARCHIVED, TRASH.

INPUT:

- `tags` – dictionary with keys usernames and values a list of tags, where a tag is a string or ARCHIVED, ACTIVE, TRASH.

set_user_view(*user*, *x*)
Set the view on this worksheet for the given user.

INPUT:

- `user` - a string
- `x` - int, one of the variables ACTIVE, ARCHIVED, TRASH in worksheet.py

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.set_user_view('admin', sagenb.notebook.worksheet.ARCHIVED)
sage: W.user_view('admin') == sagenb.notebook.worksheet.ARCHIVED
True
```

set_worksheet_that_was_published(*W*)

Set the owner and id_number of the worksheet that was published to get self.

INPUT:

- W – worksheet or 2-tuple ('owner', id_number)

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: P = nb.publish_worksheet(W, 'admin')
sage: P.worksheet_that_was_published() is W
True
```

We fake things and make it look like P published itself:

```
sage: P.set_worksheet_that_was_published(P)
sage: P.worksheet_that_was_published() is P
True
```

show_all()

x.__init__(...) initializes x; see help(type(x)) for signature

snapshot_data()

x.__init__(...) initializes x; see help(type(x)) for signature

snapshot_directory()

x.__init__(...) initializes x; see help(type(x)) for signature

start_next_comp()

x.__init__(...) initializes x; see help(type(x)) for signature

state_number()

x.__init__(...) initializes x; see help(type(x)) for signature

synchro()

x.__init__(...) initializes x; see help(type(x)) for signature

synchronize(*s*)

x.__init__(...) initializes x; see help(type(x)) for signature

system()

Return the math software system in which by default all input to the notebook is evaluated.

OUTPUT: string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('A Test Worksheet', 'admin')
sage: W.system()
'sage'
sage: W.set_system('mathematica')
```

```
sage: W.system()
'mathematica'
```

system_index()
Return the index of the current system into the Notebook's list of systems. If the current system isn't in the list, then change to the default system. This can happen if, e.g., the list changes, e.g., when changing from a notebook with Sage installed to running a server from the same directory without Sage installed. We might as well support this.

OUTPUT: integer

tags()
A temporary trivial tags implementation.

time_idle()
x.__init__(...) initializes x; see help(type(x)) for signature

time_since_last_edited()
x.__init__(...) initializes x; see help(type(x)) for signature

truncated_name(*max=30*)
x.__init__(...) initializes x; see help(type(x)) for signature

uncache_snapshot_data()
x.__init__(...) initializes x; see help(type(x)) for signature

user_autosave_interval(*username*)
x.__init__(...) initializes x; see help(type(x)) for signature

user_can_edit(*user*)
Return True if the user with given name is allowed to edit this worksheet.

INPUT:

- `user` - string

OUTPUT: bool

EXAMPLES: We create a notebook with one worksheet and two users.

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: nb.user_manager().add_user('william', 'william', 'wstein@sagemath.org', force=True)
sage: W = nb.create_new_worksheet('Test', 'sage')
sage: W.user_can_edit('sage')
True
```

At first the user 'william' can't edit this worksheet:

```
sage: W.user_can_edit('william')
False
```

After adding 'william' as a collaborator he can edit the worksheet.

```
sage: W.add_collaborator('william')
sage: W.user_can_edit('william')
True
```

Clean up:

```
sage: nb.delete()
```

user_view(*user*)

Return the view that the given user has of this worksheet. If the user currently doesn't have a view set it to ACTIVE and return ACTIVE.

INPUT:

- `user` - a string

OUTPUT:

- `Python int` - one of ACTIVE, ARCHIVED, TRASH, which are defined in worksheet.py

EXAMPLES: We create a new worksheet and get the view, which is ACTIVE:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.user_view('admin')
1
sage: sagenb.notebook.worksheet.ACTIVE
1
```

Now for the admin user we move W to the archive:

```
sage: W.move_to_archive('admin')
```

The view is now archive.

```
sage: W.user_view('admin')
0
sage: sagenb.notebook.worksheet.ARCHIVED
0
```

For any other random viewer the view is set by default to ACTIVE.

```
sage: W.user_view('foo')
1
```

user_view_is(*user*, *x*)

Return True if the user view of user is x.

INPUT:

- `user` - a string
- `x` - int, one of the variables ACTIVE, ARCHIVED, TRASH in worksheet.py

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.user_view_is('admin', sagenb.notebook.worksheet.ARCHIVED)
False
sage: W.user_view_is('admin', sagenb.notebook.worksheet.ACTIVE)
True
```

viewer_names(*max=None*)

Returns a string of the non-owner viewers on this worksheet.

INPUT:

- `max` - an integer. If this is specified, then only max number of viewers are shown.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('test1', 'admin')
sage: C = W.viewers(); C
[]
sage: C.append('sage')
sage: C.append('wstein')
sage: W.viewer_names()
'sage, wstein'
sage: W.viewer_names(max=1)
'sage, ...'
```

viewers()

Return list of viewers of this worksheet.

OUTPUT:

- `list` - of string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: nb.user_manager().add_user('hilbert','sage','sage@sagemath.org',force=True)
sage: W = nb.create_new_worksheet('test1', 'admin')
sage: W.add_viewer('hilbert')
sage: W.viewers()
['hilbert']
sage: W.add_viewer('sage')
sage: W.viewers()
['hilbert', 'sage']
```

warn_about_other_person_editing(*username*, *threshold=100*)

Check to see if another user besides username was the last to edit this worksheet during the last `threshold` seconds. If so, return True and that user name. If not, return False.

INPUT:

- `username` - user who would like to edit this file.
- `threshold` - number of seconds, so if there was no activity on this worksheet for this many seconds, then editing is considered safe.

worksheet_command(*cmd*)

x.__init__(...) initializes x; see help(type(x)) for signature

worksheet_html_filename()

Return path to the underlying plane text file that defines the worksheet.

worksheet_that_was_published()

Return a fresh copy of the worksheet that was published to get this worksheet, if this worksheet was published. Otherwise just return this worksheet.

OUTPUT: Worksheet

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Publish Test', 'admin')
sage: W.worksheet_that_was_published() is W
```

```
True
sage: S = nb.publish_worksheet(W, 'admin')
sage: S.worksheet_that_was_published() is S
False
sage: S.worksheet_that_was_published() is W
True
```

sagenb.notebook.worksheet.**Worksheet_from_basic**(*obj*, *notebook_worksheet_directory*)

INPUT:

- **obj – a dictionary (a basic Python objet) from which a** worksheet can be reconstructed.
- **notebook_worksheet_directory - string; the directory in** which the notebook object that contains this worksheet stores worksheets, i.e., nb.worksheet_directory().

OUTPUT:

- a worksheet

EXAMPLES:

```
sage: import sagenb.notebook.worksheet
sage: W = sagenb.notebook.worksheet.Worksheet('test', 0, tmp_dir(), system='gap', owner='sageuse
sage: _=W.new_cell_after(0); B = W.basic()
sage: W0 = sagenb.notebook.worksheet.Worksheet_from_basic(B, tmp_dir())
sage: W0.basic() == B
True
```

sagenb.notebook.worksheet.**after_first_word**(*s*)

Return everything after the first whitespace in the string s. Returns the empty string if there is nothing after the first whitespace.

INPUT:

- s - string

OUTPUT: a string

EXAMPLES:

```
sage: from sagenb.notebook.worksheet import after_first_word
sage: after_first_word("\%gap\n2+2\n")
'2+2\n'
sage: after_first_word("2+2")
''
```

sagenb.notebook.worksheet.**convert_time_to_string**(*t*)

Converts t (in Unix time) to a locale-specific string describing the time and date.

sagenb.notebook.worksheet.**dictify**(*s*)

INPUT:

- s - a string like 'in=5, out=7'

OUTPUT:

- dict - such as 'in':5, 'out':7

sagenb.notebook.worksheet.**extract_first_compute_cell**(*text*)

INPUT: a block of wiki-like marked up text OUTPUT:

- meta - meta information about the cell (as a dictionary)
- input - string, the input text

- `output` - string, the output text
- `end` - integer, first position after }}} in text.

`sagenb.notebook.worksheet.`**`extract_name`**(*text*)
x.__init__(...) initializes x; see help(type(x)) for signature

`sagenb.notebook.worksheet.`**`extract_system`**(*text*)
x.__init__(...) initializes x; see help(type(x)) for signature

`sagenb.notebook.worksheet.`**`extract_text_before_first_compute_cell`**(*text*)
OUTPUT: Everything in text up to the first {{{.

`sagenb.notebook.worksheet.`**`first_word`**(*s*)
Returns everything before the first whitespace in the string s. If there is no whitespace, then the entire string s is returned.

EXAMPLES:

```
sage: from sagenb.notebook.worksheet import first_word
sage: first_word("\%gap\n2+2\n")
'\\%gap'
sage: first_word("2+2")
'2+2'
```

`sagenb.notebook.worksheet.`**`format_completions_as_html`**(*cell_id*, *completions*, *username=None*)
Returns tabular HTML code for a list of introspection completions.

INPUT:

- `cell_id` - an integer or a string; the ID of the ambient cell
- `completions` - a nested list of completions in row-major order

OUTPUT:

- a string

`sagenb.notebook.worksheet.`**`ignore_prompts_and_output`**(*aString*)
Given a string s that defines an input block of code, if the first line begins in `sage:` (or >>>), strip out all lines that don't begin in either `sage:` (or >>>) or `...`, and remove all `sage:` (or >>>) and `...` from the beginning of the remaining lines.

TESTS:

```
sage: test1 = sagenb.notebook.worksheet.__internal_test1
sage: test1 == sagenb.notebook.worksheet.ignore_prompts_and_output(test1)
True
sage: test2 = sagenb.notebook.worksheet.__internal_test2
sage: sagenb.notebook.worksheet.ignore_prompts_and_output(test2)
'2 + 2\n'
```

`sagenb.notebook.worksheet.`**`next_available_id`**(*v*)
Return smallest nonnegative integer not in v.

`sagenb.notebook.worksheet.`**`split_search_string_into_keywords`**(*s*)
The point of this function is to allow for searches like this:

```
"ws 7" foo bar  Modular  '"the" end'
```

i.e., where search terms can be in quotes and the different quote types can be mixed.

INPUT:

- s - a string

OUTPUT:

- list - a list of strings

sagenb.notebook.worksheet.**update_worksheets**()
Iterate through and "update" all the worksheets. This is needed for things like wall timeouts.

sagenb.notebook.worksheet.**worksheet_filename**(*name*, *owner*)
Return the relative directory name of this worksheet with given name and owner.

INPUT:

- name - string, which may have spaces and funny characters, which are replaced by underscores.
- owner - string, with no spaces or funny characters

OUTPUT: string

EXAMPLES:

```
sage: sagenb.notebook.worksheet.worksheet_filename('Example worksheet 3', 'sage10')
'sage10/Example_worksheet_3'
sage: sagenb.notebook.worksheet.worksheet_filename('Example#%&! work\sheet 3', 'sage10')
'sage10/Example_____work_sheet_3'
```

CHAPTER

SIX

THE SAGE NOTEBOOK

AUTHORS:

- William Stein

class sagenb.notebook.notebook.**Notebook**(*dir*, *user_manager=None*)
Bases: object

active_worksheets_for(*username*)
x.__init__(...) initializes x; see help(type(x)) for signature

add_to_user_history(*entry*, *username*)
x.__init__(...) initializes x; see help(type(x)) for signature

change_worksheet_key(*old_key*, *new_key*)
x.__init__(...) initializes x; see help(type(x)) for signature

change_worksheet_name_to_avoid_collision(*worksheet*)
Change the display name of the worksheet if there is already a worksheet with the same name as this one.

color()
The default color scheme for the notebook.

conf()
x.__init__(...) initializes x; see help(type(x)) for signature

copy_worksheet(*ws*, *owner*)
x.__init__(...) initializes x; see help(type(x)) for signature

create_default_users(*passwd*)
Create the default users for a notebook.

INPUT:

- passwd - a string

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: list(sorted(nb.user_manager().users().iteritems()))
[('_sage_', _sage_), ('admin', admin), ('guest', guest), ('pub', pub)]
sage: list(sorted(nb.user_manager().passwords().iteritems())) #random
[('_sage_', ''), ('admin', ''), ('guest', ''), ('pub', '')]
sage: nb.create_default_users('newpassword')
WARNING: User 'pub' already exists -- and is now being replaced.
WARNING: User '_sage_' already exists -- and is now being replaced.
WARNING: User 'guest' already exists -- and is now being replaced.
WARNING: User 'admin' already exists -- and is now being replaced.
```

```
sage: list(sorted(nb.user_manager().passwords().iteritems())) #random
[('_sage_', ''), ('admin', ''), ('guest', ''), ('pub', '')]
sage: len(list(sorted(nb.user_manager().passwords().iteritems())))
4
```

create_new_worksheet(*worksheet_name*, *username*, *add_to_list=True*)
x.__init__(...) initializes x; see help(type(x)) for signature

create_new_worksheet_from_history(*name*, *username*, *maxlen=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

delete()
Delete all files related to this notebook.

This is used for doctesting mainly. This command is obviously *VERY* dangerous to use on a notebook you actually care about. You could easily lose all data.

EXAMPLES:

```
sage: tmp = tmp_dir(ext='.sagenb')
sage: nb = sagenb.notebook.notebook.Notebook(tmp)
sage: sorted(os.listdir(tmp))
['home']
sage: nb.delete()
```

Now the directory is gone.:

```
sage: os.listdir(tmp)
Traceback (most recent call last):
...
OSError: [Errno 2] No such file or directory: '...
```

delete_doc_browser_worksheets()
x.__init__(...) initializes x; see help(type(x)) for signature

delete_worksheet(*filename*)
Delete the given worksheet and remove its name from the worksheet list. Raise a KeyError, if it is missing.

INPUT:

- `filename` - a string

deleted_worksheets()
x.__init__(...) initializes x; see help(type(x)) for signature

empty_trash(*username*)
Empty the trash for the given user.

INPUT:

- `username` - a string

This empties the trash for the given user and cleans up all files associated with the worksheets that are in the trash.

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.new_worksheet_with_title_from_text('Sage', owner='sage')
sage: W._notebook = nb
sage: W.move_to_trash('sage')
sage: nb.worksheet_names()
```

```
['sage/0']
sage: nb.empty_trash('sage')
sage: nb.worksheet_names()
[]
```

export_worksheet(*worksheet_filename*, *output_filename*, *title=None*)
Export a worksheet, creating a sws file on the file system.

INPUT:

- `worksheet_filename` - a string e.g., 'username/id_number'
- `output_filename` - a string, e.g., 'worksheet.sws'
- **`title` - title to use for the exported worksheet (if** None, just use current title)

get_all_worksheets()
We should only call this if the user is admin!

get_server()
x.__init__(...) initializes x; see help(type(x)) for signature

get_ulimit()
x.__init__(...) initializes x; see help(type(x)) for signature

get_worksheet_with_filename(*filename*)
Get the worksheet with the given filename. If there is no such worksheet, raise a `KeyError`.

INPUT:

- `filename` - a string

OUTPUT:

- a Worksheet instance

get_worksheets_with_owner(*owner*)
x.__init__(...) initializes x; see help(type(x)) for signature

get_worksheets_with_viewer(*username*)
x.__init__(...) initializes x; see help(type(x)) for signature

html(*worksheet_filename=None*, *username='guest'*, *admin=False*, *do_print=False*)
Return the HTML for a worksheet's index page.

INPUT:

- `worksheet_filename` - a string (default: None)
- `username` - a string (default: 'guest')
- `admin` - a bool (default: False)

OUTPUT:

- a string - the worksheet rendered as HTML

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: nb.html(W.filename(), 'admin')
u'...Test...cell_input...plainclick...state_number...'
```

html_afterpublish_window(*worksheet*, *username*, *url*, *dtime*)
Return HTML for a given worksheet's post-publication page.

INPUT:

- `worksheet` - an instance of Worksheet
- `username` - a string
- `url` - a string representing the URL of the published worksheet
- `dtime` - an instance of time.struct_time representing the publishing time

OUTPUT:

- a string - the post-publication page rendered as HTML

html_beforepublish_window(*worksheet*, *username*)
Return HTML for the warning and decision page displayed prior to publishing the given worksheet.

INPUT:

- `worksheet` - an instance of Worksheet
- `username` - a string

OUTPUT:

- a string - the pre-publication page rendered as HTML

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: nb.html_beforepublish_window(W, 'admin')
u'...want to publish this worksheet?...re-publish when changes...'
```

html_download_or_delete_datafile(*ws*, *username*, *filename*)
Return the HTML for the download or delete datafile page.

INPUT:

- `username` - a string
- `ws` - an instance of Worksheet
- `filename` - a string; the name of the file

OUTPUT:

- a string - the page rendered as HTML

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: nb.html_download_or_delete_datafile(W, 'admin', 'bar')
u'...Data file: bar...DATA is a special variable...uploaded...'
```

html_edit_window(*worksheet*, *username*)
Return HTML for a window for editing `worksheet`.

INPUT:

- `username` - a string containing the username

- `worksheet` - a Worksheet instance

OUTPUT:

- a string - the editing window's HTML representation

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: nb.html_edit_window(W, 'admin')
u'...textarea class="plaintextedit"...{{{id=1|...//...}}}...'
```

html_plain_text_window (*worksheet*, *username*)

Return HTML for the window that displays a plain text version of the worksheet.

INPUT:

- `worksheet` - a Worksheet instance
- `username` - a string

OUTPUT:

- a string - the plain text window rendered as HTML

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: nb.html_plain_text_window(W, 'admin')
u'...pre class="plaintext"...cell_intext...textfield...'
```

html_share (*worksheet*, *username*)

Return the HTML for the "share" page of a worksheet.

INPUT:

- `username` - a string
- `worksheet` - an instance of Worksheet

OUTPUT:

- string - the share page's HTML representation

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: nb.html_share(W, 'admin')
u'...currently shared...add or remove collaborators...'
```

html_specific_revision (*username*, *ws*, *rev*)

Return the HTML for a specific revision of a worksheet.

INPUT:

- `username` - a string
- `ws` - an instance of Worksheet
- `rev` - a string containing the key of the revision

OUTPUT:

- a string - the revision rendered as HTML

html_upload_data_window (*ws*, *username*)
Return HTML for the "Upload Data" window.

INPUT:

- `worksheet` - an instance of Worksheet
- `username` - a string

OUTPUT:

- a string - the HTML representation of the data upload window

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: nb.html_upload_data_window(W, 'admin')
u'...Upload or Create Data File...Browse...url...name of a new...'
```

html_worksheet_revision_list (*username*, *worksheet*)
Return HTML for the revision list of a worksheet.

INPUT:

- `username` - a string
- `worksheet` - an instance of Worksheet

OUTPUT:

- a string - the HTML for the revision list

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: W = nb.create_new_worksheet('Test', 'admin')
sage: W.body()
u'\n\n{{{id=1|\n\n///\n}}}'
sage: W.save_snapshot('admin')
sage: nb.html_worksheet_revision_list('admin', W)
u'...Revision...Last Edited...ago...'
```

import_worksheet (*filename*, *owner*)
Import a worksheet with the given `filename` and set its `owner`. If the file extension is not recognized, raise a ValueError.

INPUT:

- `filename` - a string
- `owner` - a string

OUTPUT:

- `worksheet` - a newly created Worksheet instance

EXAMPLES:

We create a notebook and import a plain text worksheet into it.

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: name = tmp_filename() + '.txt'
sage: open(name,'w').write('foo\n{{{\n2+3\n}}}')
sage: W = nb.import_worksheet(name, 'admin')
```

W is our newly-created worksheet, with the 2+3 cell in it:

```
sage: W.name()
u'foo'
sage: W.cell_list()
[TextCell 0: foo, Cell 1: in=2+3, out=]
```

logout (*username*)
Do not do anything on logout (so far).

In particular, do **NOT** stop all `username`'s worksheets!

new_id_number (*username*)
Find the next worksheet id for the given user.

new_worksheet_process ()
Return a new worksheet process object with parameters determined by configuration of this notebook server.

new_worksheet_with_title_from_text (*text*, *owner*)
x.__init__(...) initializes x; see help(type(x)) for signature

pretty_print (*username=None*)
The default typeset setting for new worksheets for a given user or the whole notebook (if username is None).

TODO – only implemented for the notebook right now

pub_worksheets ()
x.__init__(...) initializes x; see help(type(x)) for signature

publish_worksheet (*worksheet*, *username*)
Publish a user's worksheet. This creates a new worksheet in the 'pub' directory with the same contents as `worksheet`.

INPUT:

- `worksheet` - an instance of Worksheet
- `username` - a string

OUTPUT:

- a new or existing published instance of Worksheet

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.load_notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('Mark','password','',force=True)
sage: W = nb.new_worksheet_with_title_from_text('First steps', owner='Mark')
sage: nb.worksheet_names()
['Mark/0']
sage: nb.create_default_users('password')
sage: nb.publish_worksheet(nb.get_worksheet_with_filename('Mark/0'), 'Mark')
pub/0: [Cell 1: in=, out=]
sage: sorted(nb.worksheet_names())
['Mark/0', 'pub/0']
```

quit()
x.__init__(...) initializes x; see help(type(x)) for signature

quit_idle_worksheet_processes()
x.__init__(...) initializes x; see help(type(x)) for signature

quit_worksheet(*W*)
x.__init__(...) initializes x; see help(type(x)) for signature

readonly_user(*username*)
Returns True if the user is supposed to only be a read-only user.

save()
Save this notebook server to disk.

save_worksheet(*W*, *conf_only=False*)
x.__init__(...) initializes x; see help(type(x)) for signature

scratch_worksheet()
x.__init__(...) initializes x; see help(type(x)) for signature

server_pool()
x.__init__(...) initializes x; see help(type(x)) for signature

set_color(*color*)
x.__init__(...) initializes x; see help(type(x)) for signature

set_not_computing()
x.__init__(...) initializes x; see help(type(x)) for signature

set_pretty_print(*pretty_print*)
x.__init__(...) initializes x; see help(type(x)) for signature

set_server_pool(*servers*)
x.__init__(...) initializes x; see help(type(x)) for signature

set_ulimit(*ulimit*)
x.__init__(...) initializes x; see help(type(x)) for signature

system(*username=None*)
The default math software system for new worksheets for a given user or the whole notebook (if username is None).

system_names()
x.__init__(...) initializes x; see help(type(x)) for signature

systems(*username=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

update_worksheet_processes()
x.__init__(...) initializes x; see help(type(x)) for signature

upgrade_model()
Upgrade the model, if needed.

- Version 0 (or non-existent model version, which defaults to 0): Original flask notebook
- Version 1: shared worksheet data cached in the User object

user(*username*)
Return an instance of the User class given the `username` of a user in a notebook.

INPUT:

- `username` - a string

OUTPUT:

- an instance of User

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().create_default_users('password')
sage: nb.user('admin')
admin
sage: nb.user('admin').get_email()
''
sage: nb.user('admin').password() #random
'256$79982100963239 79f76e9fedaf1f85bda1561c479ae732f9c1f1abab1291b0b9$373f16b9d5fab80b9a9012
```

user_history(*username*)

x.__init__(...) initializes x; see help(type(x)) for signature

user_history_text(*username*, *maxlen=None*)

x.__init__(...) initializes x; see help(type(x)) for signature

user_manager()

Returns self's UserManager object.

EXAMPLES:

```
sage: n = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: n.user_manager()
<sagenb.notebook.user_manager.OpenIDUserManager object at 0x...>
```

users_worksheets(*username*)

Returns all worksheets owned by $username$

users_worksheets_view(*username*)

Returns all worksheets viewable by $username$

valid_login_names()

Return a list of users that can log in.

OUTPUT:

- a list of strings

EXAMPLES:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.create_default_users('password')
sage: nb.valid_login_names()
['admin']
sage: nb.user_manager().add_user('Mark', 'password', '', force=True)
sage: nb.user_manager().add_user('Sarah', 'password', '', force=True)
sage: nb.user_manager().add_user('David', 'password', '', force=True)
sage: sorted(nb.valid_login_names())
['David', 'Mark', 'Sarah', 'admin']
```

worksheet(*username*, *id_number=None*)

Create a new worksheet with given id_number belonging to the user with given username, or return an already existing worksheet. If id_number is None, creates a new worksheet using the next available new id_number for the given user.

INPUT:

- `username` – string

- `id_number` - nonnegative integer or None (default)

worksheet_list_for_public(*username*, *sort='last_edited'*, *reverse=False*, *search=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

worksheet_list_for_user(*user*, *typ='active'*, *sort='last_edited'*, *reverse=False*, *search=None*)
x.__init__(...) initializes x; see help(type(x)) for signature

worksheet_names()
Return a list of all the names of worksheets in this notebook.

OUTPUT:

- a list of strings.

EXAMPLES:

We make a new notebook with two users and two worksheets, then list their names:

```
sage: nb = sagenb.notebook.notebook.Notebook(tmp_dir(ext='.sagenb'))
sage: nb.user_manager().add_user('sage','sage','sage@sagemath.org',force=True)
sage: W = nb.new_worksheet_with_title_from_text('Sage', owner='sage')
sage: nb.user_manager().add_user('wstein','sage','wstein@sagemath.org',force=True)
sage: W2 = nb.new_worksheet_with_title_from_text('Elliptic Curves', owner='wstein')
sage: nb.worksheet_names()
['sage/0', 'wstein/0']
```

class sagenb.notebook.notebook.**WorksheetDict**(*notebook*, **args*, ***kwds*)
Bases: dict

sagenb.notebook.notebook.**load_notebook**(*dir*, *interface=None*, *port=None*, *secure=None*, *user_manager=None*)
Load and return a notebook from a given directory. Create a new one in that directory, if one isn't already there.

INPUT:

- `dir` - a string that defines a directory name
- `interface` - the address of the interface the server listens at
- `port` - the port the server listens on
- `secure` - whether the notebook is secure

OUTPUT:

- a Notebook instance

sagenb.notebook.notebook.**make_path_relative**(*dir*)
Replace an absolute path with a relative path, if possible. Otherwise, return the given path.

INPUT:

- `dir` - a string containing, e.g., a directory name

OUTPUT:

- a string

sagenb.notebook.notebook.**migrate_old_notebook_v1**(*dir*)
Back up and migrates an old saved version of notebook to the new one (*sagenb*)

sagenb.notebook.notebook.**sort_worksheet_list**(*v*, *sort*, *reverse*)
Sort a given list on a given key, in a given order.

INPUT:

- `sort` - a string; ‘last_edited’, ‘owner’, ‘rating’, or ‘name’
- `reverse` - a bool; if True, reverse the order of the sort.

OUTPUT:

- the sorted list

CHAPTER

SEVEN

JAVASCRIPT (AJAX) COMPONENTS OF THE NOTEBOOK

JavaScript (AJAX) Components of the Notebook

AUTHORS:

- William Stein
- Tom Boothby
- Alex Clemesha

This file contains some minimal code to generate the Javascript code which is inserted to the head of the notebook web page. All of the interesting Javascript code is contained under `data/sage/js/notebook_lib.js`.

class `sagenb.notebook.js.`**JSKeyCode**(*key*, *alt*, *ctrl*, *shift*)

js_test()
x.__init__(...) initializes x; see help(type(x)) for signature

class `sagenb.notebook.js.`**JSKeyHandler**
This class is used to make javascript functions to check for specific keyevents.

add(*name*, *key=''*, *alt=False*, *ctrl=False*, *shift=False*)
Similar to `set_key(...)`, but this instead checks if there is an existing keycode by the specified name, and associates the specified key combination to that name in addition. This way, if different browsers don't catch one keycode, multiple keycodes can be assigned to the same test.

all_tests()
Builds all tests currently in the handler. Returns a string of javascript code which defines all functions.

set(*name*, *key=''*, *alt=False*, *ctrl=False*, *shift=False*)
Add a named keycode to the handler. When built by `all_tests()`, it can be called in javascript by `key_<key_name>(event_object)`. The function returns true if the keycode numbered by the `key` parameter was pressed with the appropriate modifier keys, false otherwise.

`sagenb.notebook.js.`**javascript**()
Return javascript library for the Sage Notebook. This is done by reading the template `notebook_lib.js` where all of the javascript code is contained and replacing a few of the values specific to the running session.

Before the code is returned (as a string), it is run through a JavascriptCompressor to minimize the amount of data needed to be sent to the browser.

The code and a hash of the code is returned.

Note: This the output of this function is cached so that it only needs to be generated once.

EXAMPLES:

```
sage: from sagenb.notebook.js import javascript
sage: s = javascript()
sage: s[0][:30]
'/* JavaScriptCompressor 0.1 [w'
```

CHAPTER

EIGHT

NOTEBOOK STYLESHEETS (CSS)

Notebook Stylesheets (CSS)

sagenb.notebook.css.**css**(*color='default'*)

Return the CSS header used by the Sage Notebook.

INPUT:

- color - string or pair of html colors, e.g., 'gmail' 'grey' ('#ff0000', '#0000ff')

EXAMPLES:

```
sage: import sagenb.notebook.css as c
sage: type(c.css()[0])
<type 'str'>
```

CHAPTER

NINE

HTML TEMPLATING FOR THE NOTEBOOK

AUTHORS:

- Bobby Moretti (2007-07-18): initial version
- Timothy Clemans and Mike Hansen (2008-10-27): major update

sagenb.notebook.template.**clean_name**(*name*)

Converts a string to a safe/clean name by converting non-alphanumeric characters to underscores.

INPUT:

•name – a string

EXAMPLES:

```
sage: from sagenb.notebook.template import clean_name
sage: print clean_name('this!is@bad+string')
this_is_bad_string
```

sagenb.notebook.template.**css_escape**(*string*)

Returns a string with all characters not legal in a css name replaced with hyphens (-).

INPUT:

•`string` – the string to be escaped.

EXAMPLES:

```
sage: from sagenb.notebook.template import css_escape
sage: css_escape('abcd')
'abcd'
sage: css_escape('12abcd')
'12abcd'
sage: css_escape(r'\'"abcd\'"')
'---abcd---'
sage: css_escape('my-invalid/identifier')
'my-invalid-identifier'
sage: css_escape(r'quotes"mustbe!escaped')
'quotes-mustbe-escaped'
```

sagenb.notebook.template.**prettify_time_ago**(*t*)

Converts seconds to a meaningful string.

INPUT

- t – time in seconds

sagenb.notebook.template.**template**(*filename*, ***user_context*)

Returns HTML, CSS, etc., for a template file rendered in the given context.

INPUT:

- `filename` - a string; the filename of the template relative to `sagenb/data/templates`
- `user_context` - a dictionary; the context in which to evaluate the file's template variables

OUTPUT:

- a string - the rendered HTML, CSS, etc.

EXAMPLES:

```
sage: from sagenb.notebook.template import template
sage: s = template(os.path.join('html', 'yes_no.html')); type(s)
<type 'unicode'>
sage: 'Yes' in s
True
sage: u = unicode('Are Gröbner bases awesome?','utf-8')
sage: s = template(os.path.join('html', 'yes_no.html'), message=u)
sage: 'Gröbner' in s.encode('utf-8')
True
```

CHAPTER

TEN

OTHER SERVERS

10.1 Sage Trac Server

This module configures and launches a Trac server, if an optional package (e.g., trac-x.y.z.spkg) is installed.

sage.server.trac.trac.**trac**(*directory='sage_trac'*, *port=10000*, *address='localhost'*, *open_viewer=False*, *auto_reload=False*, *easy_setup=False*, *options=''*)

Start a Trac server, creating a new project, if necessary. An "optional" Sage package trac-x.y.z.spkg must already be installed. (Use optional_packages() to get the exact name.)

INPUT:

- `directory` - a string (default: 'sage_trac'); name of the project directory
- `port` - an integer (default: 10000); the server's port number
- `address` - a string (default: 'localhost'); the server's address
- `open_viewer` - a bool (default: False); whether to open a browser at the server's address
- `auto_reload` - a bool (default: False); whether the server should restart automatically when *its* sources are modified
- `easy_setup` - a bool (default: False); **if** creating a new project, whether to enable optional plug-ins. See `trac_create_instance()`.
- `options` - a string (default: ''); command-line options to pass directly to tracd

sage.server.trac.trac.**trac_create_instance**(*directory='sage_trac'*, *easy_setup=False*)

Create a new Trac project instance if Trac is installed.

INPUT:

- `directory` - a string (default: 'sage_trac'); the name of the project directory
- `easy_setup` - a bool (default: False); whether to use the project name 'Sage', the default database, enable the webadmin plugin, and enable the source browser for the 'sage' Mercurial repository.

Note: To access the webadmin panel, first create a user, give that user admin permissions, and log in as that user. Create new accounts using htdigest (part of Apache):

```
cd <directory>/conf
htdigest passwd <server_address> <username>
```

Grant a user administrative privileges with:

```
trac-admin <directory> add <username> TRAC_ADMIN
```

10.2 Notebook Registration Challenges

This module includes support for challenge-response tests posed to users registering for new Sage notebook accounts. These **C**ompletely **A**utomated **P**ublic **T**uring tests to tell **C**omputers and **H**umans **A**part, or CAPTCHAs, may be simple math questions, requests for special registration codes, or reCAPTCHAs.

AUTHORS:

- reCAPTCHA is written by Ben Maurer and maintained by Josh Bronson. It is licensed under a MIT/X11 license. The reCAPTCHA challenge implemented in `reCAPTCHAChallenge` is adapted from this Python API, which is also available here.

class `sagenb.notebook.challenge.`**AbstractChallenge**(*conf*, ***kwargs*)
Bases: `object`

An abstract class with a suggested common interface for specific challenge-response schemes.

html(***kwargs*)
Returns HTML for the challenge, e.g., to insert into a new account registration page.

INPUT:

- `kwargs` - a dictionary of keywords arguments

OUTPUT:

- a string; HTML form representation of the challenge, including a field for the response, supporting hidden fields, JavaScript code, etc.

TESTS:

```
sage: from sagenb.notebook.challenge import AbstractChallenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
sage: chal = AbstractChallenge(nb.conf())
sage: chal.html()
Traceback (most recent call last):
...
NotImplementedError
```

is_valid_response(***kwargs*)
Returns the status of a challenge response.

INPUT:

- `kwargs` - a dictionary of keyword arguments

OUTPUT:

- a `ChallengeResponse` instance

TESTS:

```
sage: from sagenb.notebook.challenge import AbstractChallenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
```

```
sage: chal = AbstractChallenge(nb.conf())
sage: chal.is_valid_response()
Traceback (most recent call last):
...
NotImplementedError
```

class sagenb.notebook.challenge.**ChallengeDispatcher**(*conf*, ***kwargs*)
Bases: object

A simple dispatcher class that provides access to a specific challenge.

class sagenb.notebook.challenge.**ChallengeResponse**(*is_valid*, *error_code=None*)
Bases: object

A simple challenge response class that indicates whether a response is empty, correct, or incorrect, and, if it's incorrect, includes an optional error code.

class sagenb.notebook.challenge.**NotConfiguredChallenge**(*conf*, ***kwargs*)
Bases: sagenb.notebook.challenge.AbstractChallenge

A fallback challenge used when an administrator has not configured a specific method.

html(***kwargs*)
Returns a suggestion to inform the Notebook server's administrator about the misconfigured challenge.

INPUT:

- conf - a ServerConfiguration; an instance of the server's configuration
- kwargs - a dictionary of keyword arguments

OUTPUT:

- a string

TESTS:

```
sage: from sagenb.notebook.challenge import NotConfiguredChallenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
sage: chal = NotConfiguredChallenge(nb.conf())
sage: print chal.html()
Please ask the server administrator to configure a challenge!
```

is_valid_response(***kwargs*)
Always reports a failed response, for the sake of security.

INPUT:

- kwargs - a dictionary of keyword arguments

OUTPUT:

- a ChallengeResponse instance

TESTS:

```
sage: from sagenb.notebook.challenge import NotConfiguredChallenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
sage: chal = NotConfiguredChallenge(nb.conf())
sage: chal.is_valid_response().is_valid
```

```
False
sage: chal.is_valid_response().error_code
''
```

class sagenb.notebook.challenge.**SimpleChallenge**(*conf*, ***kwargs*)

Bases: sagenb.notebook.challenge.AbstractChallenge

A simple question and answer challenge.

html(***kwargs*)

Returns a HTML form posing a randomly chosen question.

INPUT:

- `kwargs` - a dictionary of keyword arguments

OUTPUT:

- a string; the HTML form

TESTS:

```
sage: from sagenb.notebook.challenge import SimpleChallenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
sage: chal = SimpleChallenge(nb.conf())
sage: chal.html() # random
'...What is the largest prime factor of 1001?...'
```

is_valid_response(*req_args={}*, ***kwargs*)

Returns the status of a user's answer to the challenge question.

INPUT:

- `req_args` - a string:list dictionary; the arguments of the remote client's HTTP POST request
- `kwargs` - a dictionary of extra keyword arguments

OUTPUT:

- a `ChallengeResponse` instance

TESTS:

```
sage: from sagenb.notebook.challenge import SimpleChallenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
sage: chal = SimpleChallenge(nb.conf())
sage: req = {}
sage: chal.is_valid_response(req).is_valid
sage: chal.is_valid_response(req).error_code
''
sage: from sagenb.notebook.challenge import QUESTIONS
sage: ques, ans = sorted(QUESTIONS.items())[0]
sage: ans = ans.split('|')[0]
sage: print ques
How many bits are in one byte?
sage: print ans
8
sage: req['simple_response_field'] = ans
sage: chal.is_valid_response(req).is_valid
```

```
False
sage: chal.is_valid_response(req).error_code
''
sage: req['simple_challenge_field'] = ques
sage: chal.is_valid_response(req).is_valid
True
sage: chal.is_valid_response(req).error_code
''
```

sagenb.notebook.challenge.**agree**(*response*, *answer*)

Returns whether a challenge response agrees with the answer.

INPUT:

- response - a string; the user's response to a posed challenge
- answer - a string; the challenge's right answer as a regular expression

OUTPUT:

- a boolean; whether the response agrees with the answer

TESTS:

```
sage: from sagenb.notebook.challenge import agree
sage: agree('0', r'0|zero')
True
sage: agree('eighty', r'8|eight')
False
```

sagenb.notebook.challenge.**challenge**(*conf*, ***kwargs*)

Wraps an instance of ChallengeDispatcher and returns an instance of a specific challenge.

INPUT:

- conf - a ServerConfiguration; a server configuration instance
- kwargs - a dictionary of keyword arguments

OUTPUT:

- an instantiated subclass of AbstractChallenge

TESTS:

```
sage: from sagenb.notebook.challenge import challenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
sage: nb.conf()['challenge_type'] = 'simple'
sage: chal = challenge(nb.conf())
sage: chal.html() # random
'<p>...'
```

class sagenb.notebook.challenge.**reCAPTCHAChallenge**(*conf*, *remote_ip=''*, *is_secure=False*, *lang='en'*, ***kwargs*)

Bases: sagenb.notebook.challenge.AbstractChallenge

A reCAPTCHA challenge adapted from this Python API, also hosted here, written by Ben Maurer and maintained by Josh Bronson.

html(*error_code=None*, ***kwargs*)

Returns HTML and JavaScript for a reCAPTCHA challenge and response field.

INPUT:

- `error_code` - a string (default: None); an optional error code to embed in the HTML, giving feedback about the user's *previous* response
- `kwargs` - a dictionary of extra keyword arguments

OUTPUT:

- a string; HTML and JavaScript to render the reCAPTCHA challenge

TESTS:

```
sage: from sagenb.notebook.challenge import reCAPTCHAChallenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
sage: chal = reCAPTCHAChallenge(nb.conf(), remote_ip = 'localhost')
sage: chal.html()
u'...recaptcha...'
sage: chal.html('incorrect-captcha-sol')
u'...incorrect-captcha-sol...'
```

is_valid_response(*req_args={}*, ***kwargs*)

Submits a reCAPTCHA request for verification and returns its status.

INPUT:

- `req_args` - a dictionary; the arguments of the responding user's HTTP POST request
- `kwargs` - a dictionary of extra keyword arguments

OUTPUT:

- a `ChallengeResponse` instance; whether the user's response is empty, accepted, or rejected, with an optional error string

TESTS:

```
sage: from sagenb.notebook.challenge import reCAPTCHAChallenge
sage: tmp = tmp_dir(ext='.sagenb')
sage: import sagenb.notebook.notebook as n
sage: nb = n.Notebook(tmp)
sage: chal = reCAPTCHAChallenge(nb.conf(), remote_ip = 'localhost')
sage: req = {}
sage: chal.is_valid_response(req).is_valid
sage: chal.is_valid_response(req).error_code
''
sage: req['recaptcha_response_field'] = ['subplotTimes']
sage: chal.is_valid_response(req).is_valid
False
sage: chal.is_valid_response(req).error_code
'incorrect-captcha-sol'
sage: req['simple_challenge_field'] = ['VBORw0KGgoANSUhEUgAAAB']
sage: chal.is_valid_response(req).is_valid # random
False
sage: chal.is_valid_response(req).error_code # random
'incorrect-captcha-sol'
```

CHAPTER

ELEVEN

MISCELLANEOUS

11.1 Miscellaneous Notebook Functions

TESTS:

Check that github issue #195 is fixed:

```
sage: from sagenb.misc.misc import mathjax_macros
sage: type(mathjax_macros)
<type 'list'>
```

sagenb.misc.misc.**cputime**(*t=0*)
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**encoded_str**(*obj*, *encoding='utf-8'*)
Takes an object and returns an encoded str human-readable representation.

EXAMPLES:

```
sage: from sagenb.misc.misc import encoded_str
sage: encoded_str(u'ěščřž\xfd\xe1\xed\xe9d'Ď') == 'ěščřžýáíéd'Ď'
True
sage: encoded_str(u'abc')
'abc'
sage: encoded_str(123)
'123'
```

sagenb.misc.misc.**find_next_available_port**(*interface*, *start*, *max_tries=100*, *verbose=False*)
Find the next available port at a given interface, that is, a port for which a current connection attempt returns a 'Connection refused' error message. If no port is found, raise a RuntimeError exception.

INPUT:

- `interface` - address to check
- `start` - an int; the starting port number for the scan
- `max_tries` - an int (default: 100); how many ports to scan
- `verbose` - a bool (default: True); whether to print information about the scan

OUTPUT:

- an int - the port number

EXAMPLES:

```
sage: from sagenb.misc.misc import find_next_available_port
sage: find_next_available_port('127.0.0.1', 9000, verbose=False)    # random output -- depends on
9002
```

sagenb.misc.misc.**get_languages**()
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**ignore_nonexistent_files**(*curdir*, *dirlist*)
Returns a list of non-existent files, given a directory and its contents. The returned list includes broken symbolic links. Use this, e.g., with `shutil.copytree()`, as shown below.

INPUT:

- `curdir` - a string; the name of the current directory
- `dirlist` - a list of strings; names of `curdir`'s contents

OUTPUT:

- a list of strings; names of `curdir`'s non-existent files

EXAMPLES:

```
sage: import os, shutil
sage: from sagenb.misc.misc import ignore_nonexistent_files
sage: opj = os.path.join; ope = os.path.exists; t = tmp_dir()
sage: s = opj(t, 'src'); t = opj(t, 'trg'); hi = opj(s, 'hi.txt');
sage: os.makedirs(s)
sage: f = open(hi, 'w'); f.write('hi'); f.close()
sage: os.symlink(hi, opj(s, 'good.txt'))
sage: os.symlink(opj(s, 'bad'), opj(s, 'bad.txt'))
sage: slist = sorted(os.listdir(s)); slist
['bad.txt', 'good.txt', 'hi.txt']
sage: map(lambda x: ope(opj(s, x)), slist)
[False, True, True]
sage: map(lambda x: os.path.islink(opj(s, x)), slist)
[True, True, False]
sage: shutil.copytree(s, t)
Traceback (most recent call last):
...
Error: [('.../src/bad.txt', '.../trg/bad.txt', "[Errno 2] No such file or directory: '.../src/ba
sage: shutil.rmtree(t); ope(t)
False
sage: shutil.copytree(s, t, ignore = ignore_nonexistent_files)
sage: tlist = sorted(os.listdir(t)); tlist
['good.txt', 'hi.txt']
sage: map(lambda x: ope(opj(t, x)), tlist)
[True, True]
sage: map(lambda x: os.path.islink(opj(t, x)), tlist)  # Note!
[False, False]
```

sagenb.misc.misc.**is_Matrix**(*x*)
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**open_page**(*address*, *port*, *secure*, *path=''*)
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**pad_zeros**(*s*, *size=3*)
EXAMPLES:

```
sage: pad_zeros(100)
'100'
sage: pad_zeros(10)
'010'
sage: pad_zeros(10, 5)
'00010'
sage: pad_zeros(389, 5)
'00389'
sage: pad_zeros(389, 10)
'0000000389'
```

sagenb.misc.misc.**print_open_msg**(*address*, *port*, *secure=False*, *path=''*)

Print a message on the screen suggesting that the user open their web browser to a certain URL.

INPUT:

- `address` – a string; a computer address or name
- `port` – an int; a port number
- `secure` – a bool (default: False); whether to prefix the URL with 'http' or 'https'
- `path` – a string; the URL's path following the port.

EXAMPLES:

```
sage: from sagenb.misc.misc import print_open_msg
sage: print_open_msg('localhost', 8080, True)
+-------------------------------------------------+
|                                                 |
| Open your web browser to https://localhost:8080 |
|                                                 |
+-------------------------------------------------+
sage: print_open_msg('sagemath.org', 8080, False)
+-----------------------------------------------------+
|                                                     |
| Open your web browser to http://sagemath.org:8080 |
|                                                     |
+-----------------------------------------------------+
sage: print_open_msg('sagemath.org', 90, False)
+-------------------------------------------------+
|                                                 |
| Open your web browser to http://sagemath.org:90 |
|                                                 |
+-------------------------------------------------+
sage: print_open_msg('sagemath.org', 80, False)
+-----------------------------------------------+
|                                               |
|  Open your web browser to http://sagemath.org |
|                                               |
+-----------------------------------------------+
```

sagenb.misc.misc.**register_with_cleaner**(*pid*)

x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**set_permissive_permissions**(*filename*)

x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**set_restrictive_permissions**(*filename*, *allow_execute=False*)

x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**stub**(*f*)
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**translations_path**()
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**unicode_str**(*obj*, *encoding='utf-8'*)
Takes an object and returns a unicode human-readable representation.

EXAMPLES:

```
sage: from sagenb.misc.misc import unicode_str
sage: unicode_str('ěščřžýáíéd'Ď') == u'ěščřž\xfd\xe1\xed\xe9d'Ď'
True
sage: unicode_str('abc')
u'abc'
sage: unicode_str(123)
u'123'
```

sagenb.misc.misc.**walltime**(*t=0*)
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.misc.**word_wrap**(*s*, *ncols=85*)
x.__init__(...) initializes x; see help(type(x)) for signature

11.2 Support for Notebook Introspection and Setup

AUTHORS:

- William Stein (much of this code is from IPython).
- Nick Alexander

class sagenb.misc.support.**AutomaticVariable**
Bases: sage.symbolic.expression.Expression

An automatically created symbolic variable with an additional __call__() method designed so that doing self(foo,...) results in foo.self(...).

sagenb.misc.support.**automatic_name_eval**(*s*, *globals*, *max_names=10000*)
Exec the string s in the scope of the globals dictionary, and if any NameErrors are raised, try to fix them by defining the variable that caused the error to be raised, then eval again. Try up to max_names times.

INPUT:

- s – a string
- globals – a dictionary
- max_names – a positive integer (default: 10000)

sagenb.misc.support.**automatic_name_filter**(*s*)
Wrap the string s in a call that will cause evaluation of s to automatically create undefined variable names.

INPUT:

- s – a string

OUTPUT:

- a string

sagenb.misc.support.**automatic_names**(*state=None*)

Turn automatic creation of variables and functional calling of methods on or off. Returns the current `state` if no argument is given.

This ONLY works in the Sage notebook. It is not supported on the command line.

INPUT:

- `state` – a boolean (default: None); whether to turn automatic variable creation and functional calling on or off

OUTPUT:

- a boolean, if `state` is None; otherwise, None

EXAMPLES:

```
sage: automatic_names(True)       # not tested
sage: x + y + z                   # not tested
x + y + z
```

Here, `trig_expand`, `y`, and `theta` are all automatically created:

```
sage: trig_expand((2*x + 4*y + sin(2*theta))^2)    # not tested
4*(sin(theta)*cos(theta) + x + 2*y)^2
```

IMPLEMENTATION: Here's how this works, internally. We define an `AutomaticVariable` class derived from `Expression`. An instance of `AutomaticVariable` is a specific symbolic variable, but with a special `__call__()` method. We overload the call method so that `foo(bar, ...)` gets transformed to `bar.foo(...)`. At the same time, we still want expressions like `f^2 - b` to work, i.e., we don't want to have to figure out whether a name appearing in a `NameError` is meant to be a symbolic variable or a function name. Instead, we just make an object that is both!

This entire approach is very simple—we do absolutely no parsing of the actual input. The actual real work amounts to only a few lines of code. The primary catch to this approach is that if you evaluate a big block of code in the notebook, and the first few lines take a long time, and the next few lines define 10 new variables, the slow first few lines will be evaluated 10 times. Of course, the advantage of this approach is that even very subtle code that might inject surprisingly named variables into the namespace will just work correctly, which would be impossible to guarantee with static parsing, no matter how sophisticated it is. Finally, given the target audience: people wanting to simplify use of Sage for Calculus for undergrads, I think this is an acceptable tradeoff, especially given that this implementation is so simple.

sagenb.misc.support.**completions**(*s*, *globs*, *format=False*, *width=90*, *system='None'*)

Return a list of completions in the given context.

INPUT:

- `globs` - a string:object dictionary; context in which to search for completions, e.g., `globals()`
- `format` - a bool (default: False); whether to tabulate the list
- `width` - an int; character width of the table
- `system` - a string (default: 'None'); system prefix for the completions

OUTPUT:

- a list of strings, if `format` is False, or a string

sagenb.misc.support.**cython_import**(*filename*, *verbose=False*, *compile_message=False*, *use_cache=False*, *create_local_c_file=True*)

Compile a file containing Cython code, then import and return the module. Raises an `ImportError` if anything goes wrong.

INPUT:

- `filename` - a string; name of a file that contains Cython code

OUTPUT:

- the module that contains the compiled Cython code.

`sagenb.misc.support.`**`cython_import_all`**(*filename*, *globals*, *verbose=False*, *compile_message=False*, *use_cache=False*, *create_local_c_file=True*)

Imports all non-private (i.e., not beginning with an underscore) attributes of the specified Cython module into the given context. This is similar to:

```
from module import *
```

Raises an `ImportError` exception if anything goes wrong.

INPUT:

- `filename` - a string; name of a file that contains Cython code

`sagenb.misc.support.`**`do_preparse`**()

Return True if the preparser is set to on, and False otherwise.

`sagenb.misc.support.`**`docstring`**(*obj_name*, *globs*, *system='sage'*)

Format an object's docstring to process and display in the Sage notebook.

INPUT:

- `obj_name` - a string; a name of an object
- `globs` - a string:object dictionary; a context in which to evaluate `obj_name`
- `system` - a string (default: 'sage'); the system to which to confine the search

OUTPUT:

- a string containing the object's file, type, definition, and docstring or a message stating the object is not defined

AUTHORS:

- William Stein: partly taken from IPython for use in Sage
- Nick Alexander: extensions

TESTS:

Check that Trac 10860 is fixed and we can handle Unicode help strings in the notebook:

```
sage: from sagenb.misc.support import docstring
sage: D = docstring("r.lm", globs=globals())
```

`sagenb.misc.support.`**`get_rightmost_identifier`**(*s*)

x.__init__(...) initializes x; see help(type(x)) for signature

`sagenb.misc.support.`**`help`**(*obj*)

Display HTML help for `obj`, a Python object, module, etc. This help is often more extensive than that given by 'obj?'. This function does not return a value — it prints HTML as a side effect.

Note: This a wrapper around the built-in help. If formats the output as HTML without word wrap, which looks better in the notebook.

INPUT:

- `obj` - a Python object, module, etc.

TESTS:

```
sage: import numpy.linalg
sage: import os, sage.misc.misc ; current_dir = os.getcwd()
sage: os.chdir(sage.misc.misc.tmp_dir('server_doctest'))
sage: sagenb.misc.support.help(numpy.linalg.norm)
<html><table notracebacks bgcolor="#386074" cellpadding=10 cellspacing=10><tr><td bgcolor="#f5f5
   <a target='_new' href='cell://docs-....html'>Click to open help window</a>&nbs
<br></font></tr></td></table></html>
sage: os.chdir(current_dir)
```

`sagenb.misc.support.`**`html_markup`**(*s*)
x.__init__(...) initializes x; see help(type(x)) for signature

`sagenb.misc.support.`**`init`**(*object_directory=None*, *globs={}*)
Initialize Sage for use with the web notebook interface.

`sagenb.misc.support.`**`load_session`**(*v*, *filename*, *state*)
x.__init__(...) initializes x; see help(type(x)) for signature

`sagenb.misc.support.`**`preparse_worksheet_cell`**(*s*, *globals*)
Preparse the contents of a worksheet cell in the notebook, respecting the user using `preparser(False)` to turn off the preparser. This function calls `preparse_file()` which also reloads attached files. It also does displayhook formatting by calling the `displayhook_hack()` function.

INPUT:

- `s` - a string containing code
- `globals` - a string:object dictionary; passed directly to `preparse_file()`

OUTPUT:

- a string

`sagenb.misc.support.`**`save_session`**(*filename*)
x.__init__(...) initializes x; see help(type(x)) for signature

`sagenb.misc.support.`**`setup_systems`**(*globs*)
x.__init__(...) initializes x; see help(type(x)) for signature

`sagenb.misc.support.`**`source_code`**(*s*, *globs*, *system='sage'*)
Format an object's source code to process and display in the Sage notebook.

INPUT:

- `s` - a string; a name of an object
- `globs` - a string:object dictionary; a context in which to evaluate `s`
- `system` - a string (default: 'sage'); the system to which to confine the search

OUTPUT:

- a string containing the object's file, starting line number, and source code

AUTHORS:

- William Stein: partly taken from IPython for use in Sage
- Nick Alexander: extensions

`sagenb.misc.support.`**`syseval`**(*system*, *cmd*, *dir=None*)
Evaluate an input with a "system" object that can evaluate inputs (e.g., python, gap).

INPUT:

- system - an object with an eval method that takes an input
- cmd - a string input
- sage_globals - a string:object dictionary
- dir - a string (default: None); an optional directory to change to before calling system.eval()

OUTPUT:

- system.eval() 's output

EXAMPLES:

```
sage: from sage.misc.python import python
sage: sagenb.misc.support.syseval(python, '2+4/3')
3
''
sage: sagenb.misc.support.syseval(python, 'import os; os.chdir(".")')
''
sage: sagenb.misc.support.syseval(python, 'import os; os.chdir(1,2,3)')
Traceback (most recent call last):
...
TypeError: chdir() takes exactly 1 argument (3 given)
sage: sagenb.misc.support.syseval(gap, "2+3")
'5'
```

sagenb.misc.support.**tabulate**(*v*, *width=90*, *ncols=3*)
x.__init__(...) initializes x; see help(type(x)) for signature

sagenb.misc.support.**variables**(*with_types=True*)
x.__init__(...) initializes x; see help(type(x)) for signature

11.3 Sage Notebook Introspection

TODO: - add support for grabbing source code from Pyrex functions (even if not perfect is better than nothing). - PNG or MathML output format for docstring

sagenb.misc.introspect.**introspect**(*S*, *query*, *format='html'*)
Return introspection from a given query string.

INPUT:

- S - a Sage0 object, i.e., an interface to a running instance of Python with the Sage libraries loaded
- query - a string: - if has no '?' then return completion list - if begins or ends in one '?' return docstring - if begins or ends in '??' return source code
- format - (string) 'html', 'png', 'none' (only html is implemented right now!)

11.4 This is a stand-in for Sage's inspection code in

sage.misc.sageinspect. If Sage is available, that code will be used here. Otherwise, use simple-minded replacements based on Python's inspect module.

AUTHORS:

- John Palmieri, Simon King

sagenb.misc.sageinspect.**sagenb_getdef**(*obj*, *obj_name=''*)
Return the definition header for any callable object.

INPUT:

- obj - function
- obj_name - string (optional, default '')

This calls inspect.getargspec, formats the result, and prepends obj_name.

EXAMPLES:

```
sage: from sagenb.misc.sageinspect import sagenb_getdef
sage: def f(a, b=0, *args, **kwds): pass
sage: sagenb_getdef(f, 'hello')
'hello(a, b=0, *args, **kwds)'
```

sagenb.misc.sageinspect.**sagenb_getdoc**(*obj*, *obj_name=''*)
Return the docstring associated to obj as a string. This is essentially a front end for inspect.getdoc.

INPUT: obj, a function, module, etc.: something with a docstring. If "self" is present in the docmentation, then replace it with obj_name.

EXAMPLES:

```
sage: from sagenb.misc.sageinspect import sagenb_getdoc
sage: sagenb_getdoc(sagenb.misc.sageinspect.sagenb_getdoc)[0:55]
'Return the docstring associated to ``obj`` as a string.'
```

11.5 Process docstrings with Sphinx

Processes docstrings with Sphinx. Can also be used as a commandline script:

```
python sphinxify.py <text>
```

AUTHORS:

- Tim Joseph Dumol (2009-09-29): initial version

sagenb.misc.sphinxify.**generate_configuration**(*directory*)
Generates a Sphinx configuration in directory.

INPUT:

- directory - string, base directory to use

EXAMPLES:

```
sage: from sagenb.misc.sphinxify import generate_configuration
sage: import tempfile, os
sage: tmpdir = tempfile.mkdtemp()
sage: generate_configuration(tmpdir)
sage: open(os.path.join(tmpdir, 'conf.py')).read()
'\n...extensions =...'
```

sagenb.misc.sphinxify.**is_sphinx_markup**(*docstring*)
Returns whether a string that contains Sphinx-style ReST markup.

INPUT:

- docstring - string to test for markup

OUTPUT:

•boolean

sagenb.misc.sphinxify.**sphinxify**(*docstring*, *format='html'*)

Runs Sphinx on a `docstring`, and outputs the processed documentation.

INPUT:

•`docstring` – string – a ReST-formatted docstring

•`format` – string (optional, default 'html') – either 'html' or 'text'

OUTPUT:

•string – Sphinx-processed documentation, in either HTML or plain text format, depending on the value of `format`

EXAMPLES:

```
sage: from sagenb.misc.sphinxify import sphinxify
sage: sphinxify('A test')
'\n<div class="docstring">\n    \n  <p>A test</p>\n\n\n</div>'
sage: sphinxify('**Testing**\n`monospace`')
'\n<div class="docstring"...<strong>Testing</strong>\n<span class="math"...</p>\n\n\n</div>'
sage: sphinxify('``x=y``')
'\n<div class="docstring">\n    \n  <p><span class="math">x=y</span></p>\n\n\n</div>'
sage: sphinxify('``x=y``', format='text')
'x=y\n'
sage: sphinxify(':math:`x=y`', format='text')
'x=y\n'
```

TESTS:

```
sage: n = len(sys.path)
sage: _ = sphinxify('A test')
sage: assert n == len(sys.path)
```

11.6 Live Documentation in the Notebook

Conversion of HTML (output by Sphinx or docutils) to Sage worksheet txt file.

This takes an HTML document, i.e., Sage documentation, and returns it in the editable format (notebook worksheet format with evaluable examples). It also returns a string representing the CSS link for the document. The SGML parser is setup to return only the body of the HTML documentation page and to re-format Sage examples and type-setting.

This module contains three classes:

- `sagenb.notebook.docHTMLProcessor.genericHTMLProcessor`: gathers all the common methods of the other two classes.
- `sagenb.notebook.docHTMLProcessor.SphinxHTMLProcessor`: translates HTML file generated by Sphinx into a worksheet text file
- `sagenb.notebook.docHTMLProcessor.docutilsHTMLProcessor`: translates HTML file generated by docutils `rst2html` command into a worksheet text file

---

Note: This extension of sgmllib.SGMLParser was partly inspired by Mark Pilgrim's 'Dive Into Python' examples.

AUTHORS:

- Dorian Raymer (2006): first version
- William Stein (2007-06-10): rewrite to work with twisted Sage notebook
- Mike Hansen (2008-09-27): Rewrite to work with Sphinx HTML documentation
- Sebastien Labbe (2011-01-15): Added a new class named docutilsHTMLProcessor used for translating the html output of the rst2html docutils command run on a rst file into worksheet text file. Also added a new class named genericHTMLProcessor which gathers the common methods of both docutilsHTMLProcessor and SphinxHTMLProcessor classes. Added lots of doctests to make its coverage 100% doctested.

EXAMPLES:

Process the output of docutils `rst2html` command:

```
sage: rst = ""
sage: rst += "Additions in Sage\n"
sage: rst += "-----------------\n"
sage: rst += "\n"
sage: rst += "Let's do easy computations with Sage::\n"
sage: rst += "\n"
sage: rst += "    s" + "age: 4 + 3\n"
sage: rst += "    7\n"
sage: rst += "    s" + "age: 1 - 2\n"
sage: rst += "    -1\n"
sage: rst += "\n"
sage: rst += "Let's do `x^2`::\n"
sage: rst += "\n"
sage: rst += "    s" + "age: x^2\n"
sage: rst += "    x^2\n"
sage: from docutils.core import publish_string
sage: html = publish_string(rst, writer_name='html')
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: txt = p.process_doc_html(html)
sage: len(txt)
191
sage: print txt
<h1 class="title">Additions in Sage</h1>



<p>Let's do easy computations with Sage:</p>

{{{id=0|
4 + 3
///
7
}}}

{{{id=1|
1 - 2
///
-1
}}}

<p>Let's do $x^2$:</p>

{{{id=2|
x^2
```

```
///
x^2
}}}
```

class sagenb.notebook.docHTMLProcessor.**SphinxHTMLProcessor**(*verbose=0*)

Bases: sagenb.notebook.docHTMLProcessor.genericHTMLProcessor

Initialize and reset this instance.

end_div()

Once we end the highlighted div, convert all of the pieces to cells.

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings\n']
sage: p.keep_data = True
sage: attrs = [('class', 'highlight')]
sage: p.start_div(attrs)
sage: p.start_pre([])
sage: sprompt = 'sa' + 'ge' + ': '     # to avoid problems with doctest script
sage: p.handle_data('%s4+4\n8\n%sx^2\nx^2\n' % (sprompt, sprompt))
sage: p.end_pre()
sage: p.end_div()
sage: print p.all_pieces
a lot of stuff done bunch of tmp strings
{{{id=0|
4+4
///
8
}}}

{{{id=1|
x^2
///
x^2
}}}
sage: p.temp_pieces
[]
sage: p.in_highlight_div
False

sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: attrs = [('class', 'something-else')]
sage: p.start_div(attrs)
sage: p.handle_data('some data')
sage: p.end_div()
sage: print p.all_pieces
a lot of stuff done
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<div class="something-else">', 'some data', '</div>']
sage: p.in_highlight_div
False
```

end_form()

Ignore all of the pieces since we started the form.

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.end_form()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
[]
```

end_pre()

Ignore tag </pre> when inside highligh div.

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.in_highlight_div = True
sage: p.end_pre()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']

sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.in_highlight_div = False
sage: p.end_pre()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '</pre>']
```

end_span()

Ignore all spans that occur within highlighted blocks

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.in_highlight_div = True
sage: p.end_span()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']

sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
```

```
sage: p.in_highlight_div = False
sage: p.end_span()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '</span>']
```

false_positive_input_output_cell(*cell_piece*)

Return the untouched html string of a false positive input output cell.

A false positive input-output cell come from a block of code which doesn't start with the sage prompt string 'sage:' or the Python prompt '>>>'.

INPUT:

- `cell_piece` - string, a cell piece

OUPUT:

string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: s = "sage -rst2html -h"
sage: print p.false_positive_input_output_cell(s)
<div class="highlight"><pre>
sage -rst2html -h
</pre></div>
```

reset()

Initialize necessary variables. Called by `SGMLParser.__init__()`.

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: d = SphinxHTMLProcessor()    #indirect doctest
sage: d.keep_data
False
sage: d.in_highlight_div
False
sage: d.temp_pieces
[]
sage: d.all_pieces
''
sage: d.cellcount
0
```

start_div(*attrs*)

Find out if we are starting a highlighted div.

Once we hit the tag in a highlighted block, hand of all of the pieces we've encountered so far and ignore the tag.

INPUT:

- `attrs` - list of tuple

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
```

```
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: attrs = [('class', 'highlight')]
sage: p.start_div(attrs)
sage: p.all_pieces
'a lot of stuff done bunch of tmp strings'
sage: p.temp_pieces
[]
sage: p.in_highlight_div
True

sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: attrs = [('class', 'something-else')]
sage: p.start_div(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<div class="something-else">']
sage: p.in_highlight_div
False
```

start_form(*attrs*)

Hand of everything we've accumulated so far.

Forms are ignored.

INPUT:

- `attrs` - list of tuple

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: attrs = []
sage: p.start_form(attrs)
sage: p.all_pieces
'a lot of stuff done bunch of tmp strings'
sage: p.temp_pieces
[]
```

start_pre(*attrs*)

Ignore tag <pre> when inside highligh div.

INPUT:

- `attrs` - list of tuple

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.in_highlight_div = True
```

```
sage: attrs = []
sage: p.start_pre(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']

sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.in_highlight_div = False
sage: attrs = []
sage: p.start_pre(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<pre>']
```

start_span(*attrs*)

Ignore all spans that occur within highlighted blocks

INPUT:

- attrs - list of tuple

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.in_highlight_div = True
sage: attrs = []
sage: p.start_span(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']

sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: p = SphinxHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.in_highlight_div = False
sage: attrs = []
sage: p.start_span(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<span>']
```

class sagenb.notebook.docHTMLProcessor.**docutilsHTMLProcessor**(*verbose=0*)

Bases: sagenb.notebook.docHTMLProcessor.genericHTMLProcessor

Translates output of the docutils parser rst2html into notebook text.

EXAMPLES:

```
sage: rst = ""
sage: rst += "Additions in Sage\n"
sage: rst += "-----------------\n"
sage: rst += "\n"
sage: rst += "Let's do easy computations with Sage::\n"
sage: rst += "\n"
sage: rst += "    s" + "age: 4 + 3\n"
sage: rst += "    7\n"
sage: rst += "    s" + "age: 1 - 2\n"
sage: rst += "    -1\n"
sage: rst += "\n"
sage: rst += "Let's do `x^2`::\n"
sage: rst += "\n"
sage: rst += "    s" + "age: x^2\n"
sage: rst += "    x^2\n"
sage: from docutils.core import publish_string
sage: html = publish_string(rst, writer_name='html')
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: txt = p.process_doc_html(html)
sage: len(txt)
191
sage: print txt
<h1 class="title">Additions in Sage</h1>



<p>Let's do easy computations with Sage:</p>

{{{id=0|
4 + 3
///
7
}}}

{{{id=1|
1 - 2
///
-1
}}}

<p>Let's do $x^2$:</p>

{{{id=2|
x^2
///
x^2
}}}
```

end_cite()

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.start_cite([])
```

```
sage: p.handle_data('x^2')
sage: p.end_cite()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '$', 'x^2', '$']
```

end_div()

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.end_div()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']
```

end_pre()

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: attrs = [('class', 'literal-block')]
sage: p.start_pre(attrs)
sage: sprompt = 'sa' + 'ge' + ': '    # to avoid problems with doctest script
sage: p.handle_data('%s4+4\n8\n%sx^2\nx^2\n' % (sprompt, sprompt))
sage: p.end_pre()
sage: print p.all_pieces
a lot of stuff done bunch of tmp strings
{{{id=0|
4+4
///
8
}}}

{{{id=1|
x^2
///
x^2
}}}
sage: p.temp_pieces
[]
sage: p.in_pre_litteral_block
False

sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: attrs = [('class', 'something-else')]
sage: p.start_pre(attrs)
sage: p.handle_data('some data')
```

```
sage: p.end_pre()
sage: print p.all_pieces
a lot of stuff done
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<pre class="something-else">', 'some data', '</pre>']
sage: p.in_pre_litteral_block
False
```

end_script()

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.end_script()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data
True
```

false_positive_input_output_cell(*cell_piece*)

Return the untouched html string of a false positive input output cell.

A false positive input-output cell come from a block of code which doesn't start with the sage prompt string 'sage:' or the Python prompt '>>>'.

INPUT:

- `cell_piece` - string, a cell piece

OUPUT:

string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: s = "sage -rst2html -h"
sage: print p.false_positive_input_output_cell(s)
<pre class="literal-block">
sage -rst2html -h
</pre>
```

reset()

Initialize necessary variables. Called by `SGMLParser.__init__()`.

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: d = docutilsHTMLProcessor()     #indirect doctest
sage: d.keep_data
False
sage: d.in_pre_litteral_block
False
sage: d.in_div_footer_block
False
sage: d.temp_pieces
```

```
[]
sage: d.all_pieces
''
sage: d.cellcount
0
```

start_cite(*attrs*)

INPUT:

- attrs - list of tuple

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: attrs = []
sage: p.start_cite(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '$']
```

start_div(*attrs*)

INPUT:

- attrs - list of tuple

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: attrs = [('class', 'document'), ('id', 'title')]
sage: p.start_div(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']
```

start_pre(*attrs*)

INPUT:

- attrs - list of tuple

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: attrs = [('class', 'literal-block')]
sage: p.start_pre(attrs)
sage: p.all_pieces
'a lot of stuff done bunch of tmp strings'
sage: p.temp_pieces
[]
```

```
sage: p.in_pre_litteral_block
True

sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: attrs = [('class', 'something-else')]
sage: p.start_pre(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<pre class="something-else">']
sage: p.in_pre_litteral_block
False
```

start_script(*attrs*)

INPUT:

- attrs - list of tuple

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: attrs = [('type', 'text/x-mathjax-config')]
sage: p.start_script(attrs)
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data
False
```

class sagenb.notebook.docHTMLProcessor.**genericHTMLProcessor**(*verbose=0*)

Bases: sgmllib.SGMLParser

This class gathers the methods that are common to both classes sagenb.notebook.SphinxHTMLProcessor and sagenb.notebook.docutilsHTMLProcessor.

end_body()

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.end_body()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']
```

end_html()

INPUT:

- data - string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.end_html()
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings']
```

get_cellcount()

Return the current cell count and increment it by one.

OUTPUT:

- an int

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: d = docutilsHTMLProcessor()
sage: d.get_cellcount()
0
sage: d.get_cellcount()
1

sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: d = SphinxHTMLProcessor()
sage: d.get_cellcount()
0
sage: d.get_cellcount()
1
```

hand_off_temp_pieces(*piece_type*)

To separate the documentation's content from the Sage examples, everything is split into one of two cell types. This method puts the current `self.temp_pieces` into `self.all_pieces`.

INPUT:

- `piece_type` - a string; indicates the type of and how to process the current `self.temp_pieces`. It can be one of the following:
  - `"to_doc_pieces"` - put temp_pieces in all_pieces
  - `"ignore"` - delete temp_pieces
  - `"to_cell_pieces"` - translate temp_pieces into cells and put it in all_pieces

EXAMPLES:

Move temporary pieces to all pieces:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.hand_off_temp_pieces('to_doc_pieces')
sage: p.all_pieces
'a lot of stuff done bunch of tmp strings'
sage: p.temp_pieces
[]
```

Ignore temporary pieces:

```
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.hand_off_temp_pieces('ignore')
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
[]
```

Translate temporary pieces (starting with sage prompt) into cells:

```
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['sage'+': 4+4\n', '8\n', 'sage'+': 9-4\n', '5\n']
sage: p.hand_off_temp_pieces('to_cell_pieces')
sage: print p.all_pieces
a lot of stuff done
{{{id=0|
4+4
///
8
}}}

{{{id=1|
9-4
///
5
}}}
sage: p.temp_pieces
[]
```

Translate temporary pieces (not starting with sage prompt) into cells:

```
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.hand_off_temp_pieces('to_cell_pieces')
sage: print p.all_pieces
a lot of stuff done <pre class="literal-block">
bunch of tmp strings
</pre>
sage: p.temp_pieces
[]
```

handle_charref(*ref*)

INPUT:

- `ref` - string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.handle_charref('160')
sage: p.all_pieces
'a lot of stuff done '
```

```
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', ' ']
```

handle_comment(*data*)

INPUT:

- data - string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.handle_comment('important comment')
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<!--important comment-->']
```

handle_data(*data*)

INPUT:

- data - string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.handle_data('some important data')
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', 'some important data']
```

handle_decl(*text*)

INPUT:

- data - string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.handle_decl('declaration')
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<!declaration>']
```

handle_entityref(*ref*)

INPUT:

- ref - string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.handle_entityref('160')
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '&160']
```

handle_pi (*text*)

Handle processing instructions

INPUT:

- `text` - string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.handle_pi('instructions')
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<?instructions>']
```

process_cell_input_output (*cell_piece*)

Process and return a `cell_piece`.

All divs with CSS class="highlight" (if generated with Sphinx) or class="literal-block" (if generated with docutils) contain code examples. They include

- Models of how the function works. These begin with, e.g., 'INPUT:' and are re-styled as divs with class="usage_model".
- Actual Sage input and output. These begin with 'sage:'. The input and output are separated according to the Notebook edit format.

INPUT:

- `cell_piece` - a string; a cell piece

OUTPUT:

- a string; the processed cell piece

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: s = "s" + "age: 4 + 4\n8"    # avoid the doctest script to parse "sage:"
sage: p.process_cell_input_output(s)
'\n{{{id=0|\n4 + 4\n///\n8\n}}}\n\n'
sage: print p.process_cell_input_output(s)
{{{id=1|
4 + 4
```

```
///
8
}}}

sage: s = "age: 4 + 4\n8"
sage: print p.process_cell_input_output(s)
<pre class="literal-block">
age: 4 + 4
8
</pre>

sage: s = '&gt;'*3 + " 4 + 4\n8"
sage: print p.process_cell_input_output(s)
{{{id=2|
4 + 4
///
8
}}}

sage: s = "s" + "age: 4 + 4\n8\ns" + "age: 2 + 2\n4"
sage: print p.process_cell_input_output(s)
{{{id=3|
4 + 4
///
8
}}}

{{{id=4|
2 + 2
///
4
}}}
```

process_doc_html(*doc_in*)

Returns processed HTML input as HTML output. This is the only method that needs to be called externally.

INPUT:

- doc_in - a string containing properly formed HTML

OUTPUT:

- a string; the processed HTML

EXAMPLES:

```
sage: rst = ""
sage: rst += "Title\n"
sage: rst += "-----\n"
sage: rst += "n"
sage: rst += "Some text\n"
sage: from docutils.core import publish_string
sage: html = publish_string(rst, writer_name='html')
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: txt = p.process_doc_html(html)
sage: len(txt)
51
sage: txt
'<h1 class="title">Title</h1>\n\n<p>nSome text</p>\n\n\n\n'
```

start_body(*attrs*)

Set `self.keep_data` to True upon finding the opening body tag.

INPUT:

- `attrs` - a string:string dictionary containing the element's attributes

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import SphinxHTMLProcessor
sage: d = SphinxHTMLProcessor()
sage: d.keep_data
False
sage: d.start_body(None)
sage: d.keep_data
True

sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: d = docutilsHTMLProcessor()
sage: d.keep_data
False
sage: d.start_body(None)
sage: d.keep_data
True
```

unknown_endtag(*tag*)

INPUT:

- `tag` - string

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: p.unknown_endtag('head')
sage: p.all_pieces
'a lot of stuff done '
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '</head>']
```

unknown_starttag(*tag*, *attrs*)

INPUT:

- `tag` - string
- `attrs` - list of tuples

EXAMPLES:

```
sage: from sagenb.notebook.docHTMLProcessor import docutilsHTMLProcessor
sage: p = docutilsHTMLProcessor()
sage: p.all_pieces = 'a lot of stuff done '
sage: p.temp_pieces = ['bunch ', 'of ', 'tmp ', 'strings']
sage: p.keep_data = True
sage: tag = 'style'
sage: attrs = [('type', 'text/css')]
sage: p.unknown_starttag(tag, attrs)
sage: p.all_pieces
'a lot of stuff done '
```

```
sage: p.temp_pieces
['bunch ', 'of ', 'tmp ', 'strings', '<style type="text/css">']
```

CHAPTER

TWELVE

STORAGE

12.1 Sage Notebook Storage Abstraction Layer

class sagenb.storage.abstract_storage.**Datastore**
Bases: object

The Sage Notebook storage abstraction layer abstract base class. Each storage abstraction layer derives from this.

create_worksheet(*username*, *id_number*)
Create worksheet with given id_number belonging to the given user.

If the worksheet already exists, return ValueError.

INPUT:

- username – string
- id_number – integer

OUTPUT:

- a worksheet

delete()
Delete all files associated with this datastore. Dangerous! This is only here because it is useful for doctesting.

export_worksheet(*username*, *id_number*, *filename*, *title*)
Export the worksheet with given username and id_number to the given filename (e.g., 'worksheet.sws').

INPUT:

- **title - title to use for the exported worksheet (if** None, just use current title)

import_worksheet(*username*, *id_number*, *filename*)
Input the worksheet username/id_number from the file with given filename.

load_openid()
x.__init__(...) initializes x; see help(type(x)) for signature

load_server_conf()
x.__init__(...) initializes x; see help(type(x)) for signature

load_user_history(*username*)
Return the history log for the given user.

INPUT:

- `username` – string

OUTPUT:

- list of strings

load_users()
OUTPUT:

- dictionary of user info

load_worksheet(*username*, *id_number*)
Return worksheet with given id_number belonging to the given user.

If the worksheet does not exist, return ValueError.

INPUT:

- `username` – string
- `id_number` – integer

OUTPUT:

- a worksheet

save_openid(*openid_dict*)
x.__init__(...) initializes x; see help(type(x)) for signature

save_server_conf(*server_conf*)
x.__init__(...) initializes x; see help(type(x)) for signature

save_user_history(*username*, *history*)
Save the history log (a list of strings) for the given user.

INPUT:

- `username` – string
- `history` – list of strings

save_users(*users*)
INPUT:

- `users` – dictionary mapping user names to users

save_worksheet(*worksheet*, *conf_only=False*)
INPUT:

- `worksheet` – a Sage worksheet
- `conf_only` – default: False; if True, only save the config file, not the actual body of the worksheet

worksheets(*username*)
Return list of all the worksheets belonging to the user with given name. If the given user does not exists, an empty list is returned.

EXAMPLES: The load_user_data function must be defined in the derived class:

```
sage: from sagenb.storage.abstract_storage import Datastore
sage: Datastore().worksheets('foobar')
Traceback (most recent call last):
...
NotImplementedError
```

12.2 A Filesystem-based Sage Notebook Datastore

Here is the filesystem layout for this datastore. Note that the all of the pickles are pickles of basic Python objects, so can be unpickled in any version of Python with or without Sage or the Sage notebook installed. They are also not compressed, so are reasonably easy to read ASCII.

The filesystem layout is as follows. It mirrors the URL's used by the Sage notebook server:

```
sage_notebook.sagenb
     conf.pickle
     users.pickle
     openid.pickle (optional)
     readonly.txt (optional)
     home/
         username0/
            history.pickle
            id_number0/
                worksheet.html
                worksheet_conf.pickle
                cells/
                data/
                snapshots/
            id_number1/
                worksheet.html
                worksheet_conf.pickle
                cells/
                data/
                snapshots/
            ...
         username1/
         ...
```

class sagenb.storage.filesystem_storage.**FilesystemDatastore**(*path*)

Bases: sagenb.storage.abstract_storage.Datastore

INPUT:

- path – string, path to this datastore

EXAMPLES:

```
sage: from sagenb.storage import FilesystemDatastore
sage: FilesystemDatastore(tmp_dir())
Filesystem Sage Notebook Datastore at ...
```

create_worksheet(*username*, *id_number*)

Create worksheet with given id_number belonging to the given user.

If the worksheet already exists, return ValueError.

INPUT:

- username – string
- id_number – integer

OUTPUT:

- a worksheet

delete()
Delete all files associated with this datastore. Dangerous! This is only here because it is useful for doctesting.

export_worksheet(*username*, *id_number*, *filename*, *title*)
Export the worksheet with given username and id_number to the given filename (e.g., 'worksheet.sws').

INPUT:

- **title - title to use for the exported worksheet (if** None, just use current title)

import_worksheet(*username*, *id_number*, *filename*)
Import the worksheet username/id_number from the file with given filename.

load_openid()
Loads an open_id dict read from the disk.

load_server_conf()
x.__init__(...) initializes x; see help(type(x)) for signature

load_user_history(*username*)
Return the history log for the given user.

INPUT:

- username – string

OUTPUT:

- list of strings

load_users(*user_manager*)
OUTPUT:

- dictionary of user info

EXAMPLES:

```
sage: from sagenb.notebook.user import User
sage: from sagenb.notebook.user_manager import SimpleUserManager
sage: U = SimpleUserManager()
sage: users = {'admin':User('admin','abc','a@b.c','admin'), 'wstein':User('wstein','xyz','b@
sage: from sagenb.storage import FilesystemDatastore
sage: ds = FilesystemDatastore(tmp_dir())
sage: ds.save_users(users)
sage: 'users.pickle' in os.listdir(ds._path)
True
sage: users = ds.load_users(U)
sage: U.users()
{'admin': admin, 'wstein': wstein}
```

load_worksheet(*username*, *id_number*)
Return worksheet with given id_number belonging to the given user.

If the worksheet does not exist, return ValueError.

INPUT:

- username – string
- id_number – integer

OUTPUT:

- a worksheet

readonly_user(*username*)
Each line of the readonly file has a username.

save_openid(*openid_dict*)
Saves an open_id dict to the disk.

save_server_conf(*server_conf*)
INPUT:

- `server` –

save_user_history(*username*, *history*)
Save the history log (a list of strings) for the given user.

INPUT:

- `username` – string
- `history` – list of strings

save_users(*users*)
INPUT:

- `users` – dictionary mapping user names to users

EXAMPLES:

```
sage: from sagenb.notebook.user import User
sage: from sagenb.notebook.user_manager import SimpleUserManager
sage: U = SimpleUserManager()
sage: users = {'admin':User('admin','abc','a@b.c','admin'), 'wstein':User('wstein','xyz','b@
sage: from sagenb.storage import FilesystemDatastore
sage: ds = FilesystemDatastore(tmp_dir())
sage: ds.save_users(users)
sage: 'users.pickle' in os.listdir(ds._path)
True
sage: users = ds.load_users(U)
sage: U.users()
{'admin': admin, 'wstein': wstein}
```

save_worksheet(*worksheet*, *conf_only=False*)
INPUT:

- `worksheet` – a Sage worksheet
- `conf_only` – default: False; if True, only save the config file, not the actual body of the worksheet

EXAMPLES:

```
sage: from sagenb.notebook.worksheet import Worksheet
sage: tmp = tmp_dir()
sage: W = Worksheet('test', 2, tmp, system='gap', owner='sageuser')
sage: from sagenb.storage import FilesystemDatastore
sage: DS = FilesystemDatastore(tmp)
sage: DS.save_worksheet(W)
```

worksheets(*username*)
Return list of all the worksheets belonging to the user with given name. If the given user does not exists, an empty list is returned.

EXAMPLES: The load_user_data function must be defined in the derived class:

```
sage: from sagenb.storage import FilesystemDatastore
sage: tmp = tmp_dir()
```

```
sage: FilesystemDatastore(tmp).worksheets('foobar')
[]
sage: from sagenb.notebook.worksheet import Worksheet
sage: W = Worksheet('test', 2, tmp, system='gap', owner='sageuser')
sage: from sagenb.storage import FilesystemDatastore
sage: DS = FilesystemDatastore(tmp)
sage: DS.save_worksheet(W)
sage: DS.worksheets('sageuser')
[sageuser/2: [Cell 0: in=, out=]]
```

sagenb.storage.filesystem_storage.**is_safe**(*a*)

Used when importing contents of various directories from Sage worksheet files. We define this function to avoid the possibility of a user crafting fake sws file such that extracting it creates files outside where we want, e.g., by including .. or / in the path of some file.

CHAPTER

THIRTEEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

S

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W