Assignment 6 - IPython clone

Deadline: Nov 29th, 23:59

Important: The scripts of this assignment should be saved in a subdirectory named `assignment6` of your INF3331 github repository.

Points will be given on correctness of the solutions, the documentation and the code readibility.

6.0 Introduction

The goal of this assignment is to develop an IPython clone. This clone will have the core functionality of IPython, such as magic functions, tab completion and will even come with a simple web interface. To this end, the script `getchar.py` is provided to simplify character input for the console interface. It can both be used to explore the behavior of different special characters, by running

```
python getchar.py
```

and imported in your solution with

```
from getchar import getchar
```

For code documentation, you should follow the Sphinx documentation guidelines.

Here is an example of a properly documented Python function:

```
def square_root(n):
    """Calculate the square root of a number.

    Args:
        n (int): the number to get the square root of.
    Returns:
        the square root of n.
    Raises:
        TypeError: if n is not a number.
        ValueError: if n is negative.

    Example usage:
    >>> print sqrt(4)
    2
    """

    return sqrt(n);
```

More full examples can be viewed here.

Remark: Exercise 6.7 (web-interface) depends only on 6.1 and the command history of 6.5.

6.1 Feed line to Python (5 points)

Create a function `feedline` that takes a single string as input and runs the string as a Python command. For this this version, the functionality of the clone can be quite basic: it should allow to import modules, create variables and functions. The return value should be a string containing the output of the command run, and the next prompt for the command after.

Here is an example usage:

```
>>> print feedline('print "hello world"')
Hello world
in [1]:
>>> print feedline("") # blank feeds does not increment the line number
in [1]:
>>> print feedline("x = 1")
in [2]:
>>> print feedline("x += 1")
in [3]:
>>> print repr(feedline("print x")) # more detail
"2\nin [4]: "
>>> print feedline("from math import sin")
in [5]:
>>> print feedline("def f(x): return sin(x**2)")
In [6]:
>>> print feedline("f(x)")
Out [7]: -0.756802495308
In [7]:
>>>
```

Hint: Try using `eval` for evaluation. If that raises an exception, use `exec`. Also, to not clutter the programs name space, create a copy of namespace for your program: `namespace = vars().copy()`. Use this as second argument when running `eval` and `exec` to ensure that the user namespace is seperated from the program namespace.

Another hint: The output of `print` statements (e.g. through `exec`) will by default be streamed to stdout (i.e. to the terminal). In our case we want to capture the output to a string so that the feedline function can return it. You achieve this with the `StringIO` module. Here is a code example:

```python
import sys
from StringIO import StringIO

namespace = vars().copy()

code = """
import random
print "hello world"
"""

# Swap stdout with a StringIO instance
oldio, sys.stdout = sys.stdout, StringIO()
exec(code, namespace)
# Get stdout buffer
out = sys.stdout.getvalue()
# Reset stdout
sys.stdout = oldio

# Print out captured stdout
print out
```

File: `feedline.py`

6.2 Create basic interface (4 points)

Create a function `prompt` that initiates a basic python interface. Each character should be displayed as the user types, but also buffered, such that when the newline-character is typed (either `\n` or `\r` depending on system), the characters in the buffer is passed to the `feedline` function. Initiate this function if `__name__ == "__main__"`.

Here is an example usage:

```
$ python mypython.py
Welcome to mypython!
in [0]: print "Hello world"
Hello world
in [1]:
```

Note: `getchar` (download on INF3331 webpage) intercepts all characters including escape characters for stopping your program. Be careful not to get caught in an infinity loop that you can not get out of while make this program.

Note also, `print` adds newlines (or spaces) to the output. For a more controlled output, use the function `sys.stdout.write()`.

File: `mypython.py`

6.3 Error handling (2 points)

Extend the solution of 6.1 (`feedline.py`) to handle errors gracefully, such as syntax errors or execution exceptions. In case of an error, the `feedline` function should return a description of the error and a traceback of the program. The program should not terminate if not specifically ask to.

A minimal solution could look as follows:

```
Welcome to mypython!
in [0]: not_a_variable
Error: name 'not_a_variable' is not defined
in [1]: not_an_array = 42
in [2]: not_an_array[0]
Error: 'int' object has no attribute '__getitem__'
in [3]:
```

Tip: use `try, except`, and either `traceback.print_exc(file=sys.stdout)` from the `traceback` module or `sys.exc_info`.

File: `feedline.py`

6.4 Control sequence (3 points)

When pressing CTRL-D on the input, `mypython` should have one of two behaviors: If there are characters in the buffer, empty the buffer and present the user with a new prompt. If however the buffer already is empty, exit without an error. Therefore, a double tap on CTRL-D should result in the session to exit if there are characters in the buffer. Add this feature to `mypython.py`

```
$ python mypython.py
Welcome to mypython!
in [0]: asdf            # press 'asdf' + CTRL-D
KeyboardInterupt
in [0]:                 # press CTRL-D only
kthankxbye!
$
```

Tip: Run `python getchar.py` to identify the code for `CTRL-D`.

6.5 Up and down characters and command history (4 points)

Each time a line is fed into `feedline`, a (local variable) list with all the history of all commands is updated. Add this feature (to `mypython.py` or `feedline.py`).

When this is implemented, also implement triggers for the up and down button such that the user can access the history.

```
$ python mypython.py
Welcome to mypython!
in [0]: print 1                # press 'print 1\n'
1
in [1]: print 2                # press 'print 2\n'
2
in [2]:                        # press up twice, then '\n'
in [2]: print 2                # preferably on same line
in [2]: print 1                # but new lines for each iteration is allowed.
1
in [3]: print 3                # press 'print 3' + up + down + '\n'
in [3]: print 1
in [3]: print 3
3
```

Hint: getchar only retrieves a single byte for each character. The arrow-keys uses multiple bytes in their representation. All the arrow keys should have the same first unique key prefix that getchar can retrieve as normal. Then use `sys.stdin.read(2)` to retrieve the missing characters that identify arrow direction.

6.6 Magic commands (4 points)

Extend the implementation with the following magic commands (as in ipython):

- If a line is prefixed with an exclamation mark '!', the command should be passed to the operating system and the STDOUT and STDERR be displated to screen. Example:

  ```
  $ python mypython.py
  Welcome to mypython!
  in [0]: !echo 123
  123
  in [1]:
  ```

  Hint: Use the module `subprocess.Popen` or `os.system` to pass commands to the operating system.

- `object?` to display the docstring of `object`. Hint: Use `help(object)` or `inspect.getdoc`.

- `%save filename` to save all input lines of that interactive session of the user to `filename`. Tip: use the command history from 6.5.

File: `feedline.py`

6.7 Webinterface (8 points)

Build a simple webinterface of the IPython clone. We recommend the Flask web application framework for this exercise, but you may choose a different framework if you like.

The top of the webinterface should show a list of all previous `mypython` inputs and outputs (if there are any), similar to the console version.

The bottom of the web interface should have a form consisting of a text input field where the user can type in python commands, and a submit button. Pressing the submit button should send a `POST` request to the server with the python command, which should then be passed to the `feedline` function from 6.1, followed by a re-rendering of the webpage.

Tip: You can pass lists to flask templates and iterate over them in the template.

File: `mypython_web.py` and template files

6.8 Tab completion (INF4331 students only, 8 points)

Implement tab completion when the `TAB` charracter is pressed. Iterate backwards in the buffer to retrieve the name so far. (Only look at character valid in python names `r'[\w\d_]+'`.) Iterate through the namespace for partial name matches. If a single match is found, complete the word. If multiple matches are found, list the options.

Example usage:

```
$ python mypython.py
Welcome to mypython!
in [0]: aa,ab,acc = 1,2,3
in [1]: a                      # press 'a\t'
aa  ab  acc
in [1]: ac                     # press 'c\t\n'
in [1]: acc                    # this part should preferably be on same line
out [1]: 3
in [2]:
```

File: `mypython.py`

6.9 Backspace (INF4331 students only, 2 points)

Pressing the backspace button should remove a character from the display (and update any interal buffers.) Extend 6.2 such that this feature is supported.

Tip: Use `sys.stdout.write("\b")` to move the character cursor one to the left (but not overwrite the character). Note that this feature is not visible on the screen in Windows, but the feature should be possible to implement nontheless.

File: `mypython.py`

Total points (INF3331): 30 points

Total points (INF4331): 40 points