Alternative Python Interactive Shells

This is just a quick introduction to some different alternatives to the standard Python REPL that can make life a little easier for you as a developer. They are especially helpful when you need to do a bunch of experimentation in Python where it makes more sense to use an interactive interpreter than to put code in files for testing. It's beyond the scope of the class to go into detail on these alternative shells, but I hope you will look into using them.

Note: I changed my computer's command line prompt to just be `$` to not clutter the images, but I *am* in a virtual environment. **You should activate your virtual environment before installing these packages.**

It's worthwhile to play around with these alternative Python shells. If you are interested in learning more, google the name with "tutorial" or "examples", etc.

bpython

The bpython interpreter is a lightweight replacement for the REPL that provides syntax-highlighting, auto-indent, and tab-completion with suggestions.

To install:

```
$ pip install bpython
```

Here, I entered a function and as I was typing this text in, bpython automatically indents the lines when it is aware of starting/continuing a code-block. As I entered the `print_doubles(`, it displays a prompt with autocomplete suggestions.

```
$ bpython
bpython version 0.16 on top of Python 3.6.2 /Users/diane/.virtualenvs/ucsd2/bin/python3.6
>>> numbers = [2, 4, 6, 8]
>>> def print_doubles(nums):
...     for n in nums:
...         print("{}, {}".format(n, n * 2))
...
>>> print_doubles(
 print(         print_doubles(
```

The text in black is what I have typed previously (because I messed up the screen capture and had to repeat this), so if I want that, I can press the right-arrow and it accepts what it has suggested, and gives me more prompt information about the function:

```
>>> print_doubles(
 print_doubles: (nums)
```

When I type `numbers.`, with the dot, then it shows me all of the methods that are available for lists, since `numbers` is a list. I can use the tab key repeatedly to cycle through the suggestions.

```
>>> numbers.index
```

```
append          clear           copy            count
extend          index           insert          pop
remove          reverse         sort
```

Once I decide on something, I only need to start the continuation by typing a `(`, and now I get the summary documentation for the method I have chosen:

```
>>> numbers.insert(
```

```
numbers.insert: (index, object)
insert
L.insert(index, object) -- insert object before index
```

When I start to type something that it recognizes from a previous command (as you saw above), it also puts it out as a suggestion:

```
>>> print_doubles(numbers)
```

```
print(          print_doubles(
```

I can press the right arrow to accept the suggestion in black or if I press the tab key at this point, it cycles through the other suggestions.

There are other nice features of bpython; you can check the documentation to see them and decide if they are for you. Sometimes the insistent auto-suggestions are annoying, although after a while you expect them and don't notice as much. But I like the color-coding in bpython.

ipython

Probably the most popular interpreter replacement is ipython, which operates a bit differently than other interpreters, in that it has the concept of cells. Usually, each statement or code-block goes into one cell. It has some autocomplete (using the tab key) and auto-indent, as you might expect, but it doesn't have the advanced suggestions that bpython has.

To install:

```
$ pip install ipython
```

Here, as I'm typing the code in, it knows whether the next line should be indented, and indents it automatically for me, the same as bpython:

```
$ ipython
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017, 20:11:06)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: numbers = [2, 4, 6, 8]

In [2]: def print_doubles(nums):
   ...:     for n in nums:
   ...:
```

Each "cell" is numbered - so far I've used cells 1 and 2.

Here I've finished entering all the code for my function in cell 2, and now it's ready for me to enter something in cell 3.

```
In [2]: def print_doubles(nums):
   ...:     for n in nums:
   ...:         print("n: {}, doubld: {}".format(n, n * 2))
   ...:

In [3]:
```

But I have a typo! In the regular REPL, you would have to carefully use up-arrow to get the first 2 lines of the function, then when you get to the third line, edit it and then continue. If the function was a long one, this can be tedious and painful! But in ipython, pressing up-arrow gets me the *whole* code block in the previous cell, and I can edit it to my heart's content until I like it, and then continue by moving my cursor to the last line of the cell and pressing "Enter". Then I can continue.

```
In [3]: def print_doubles(nums):
   ...:     for n in nums:
   ...:         print("n: {}, double: {}".format(n, n * 2))
   ...:

In [4]: print_doubles(numbers)
n: 2, double: 4
n: 4, double: 8
n: 6, double: 12
n: 8, double: 16
```

The nice part of this is that I can change a cell anytime. Suppose I decide to change the print output formatting. I can change the function again and now it is different. Note it always advances the cell number.

```
In [5]: def print_doubles(nums):
   ...:     for n in nums:
   ...:         print("number: {}, double: {}".format(n, n * 2))
   ...:

In [6]: print_doubles(numbers)
number: 2, double: 4
number: 4, double: 8
number: 6, double: 12
number: 8, double: 16
```

ptpython and ptipython

The ptpython interpreter is very similar to the standard REPL, but includes some other features, like a vim and emacs mode.

To install:

```
$ pip install ptpython
```

Here I have used a small window to start ptpython so you can see the emacs line at the bottom of the screen.

```
$ ptpython
>>>

 [F4] Emacs  28/28 [F3] History [F6] Paste mode                [F2] Menu - CPython 3.6.2
```

Not being a user of emacs anymore, I couldn't tell you how it works; you'll have to take a look at the documentation to see how to use it. Some people swear by it, so it must have some good features.

One feature of ptpython is that if you have it installed with ipython, it has another interface, ptipython, that gives you the ptpython interface inside the ipython shell.

```
$ ptipython
Python 3.6.2 (v3.6.2:5fd33b5926, Jul 16 2017, 20:11:06)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.


In [1]:

 [F4] Emacs  30/30 [F3] History [F6] Paste mode                [F2] Menu - CPython 3.6.2
```

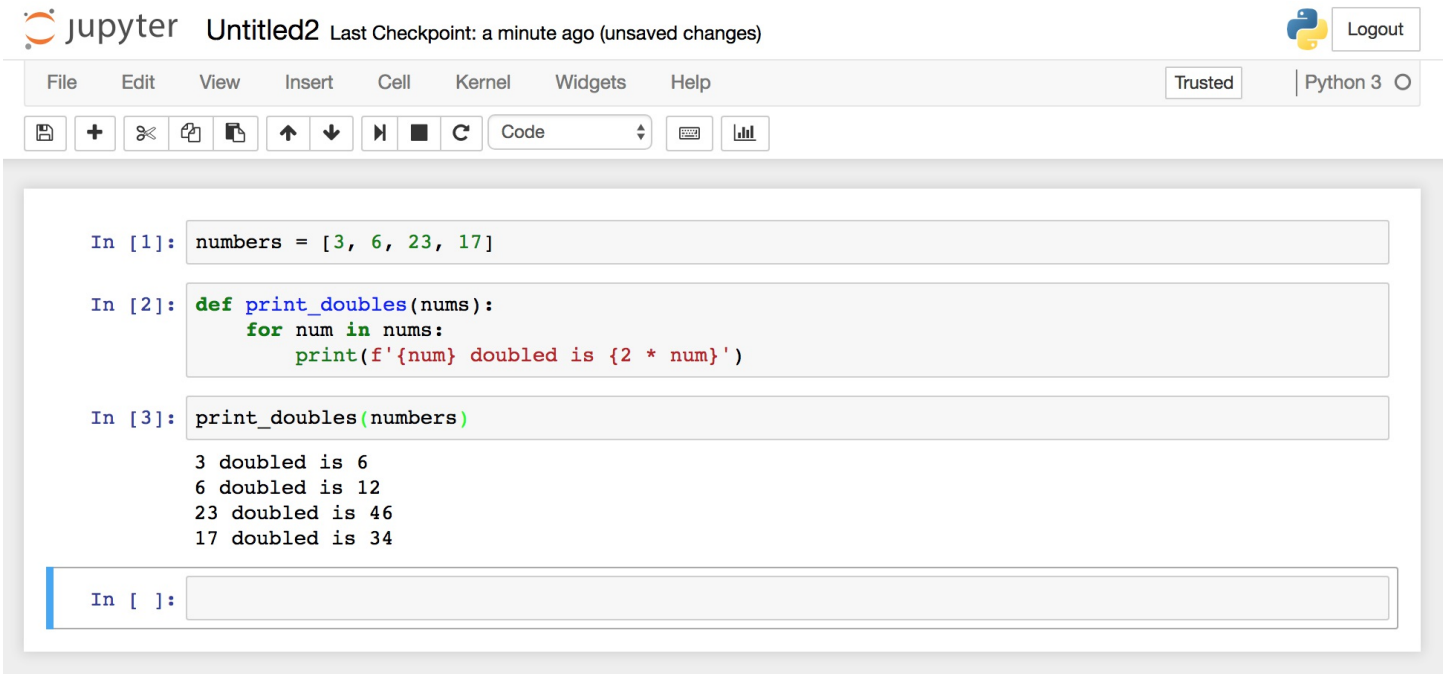
Anaconda and Jupyter Notebooks

And now for something completely different! Not really a REPL alternative, but a whole system for using Python.

In the fields of BioInformatics, Data Science, and scientific computing, among others, the Anaconda distribution of Python is arguably the most popular. It includes many scientific packages for fast number-crunching, data science, visualization, machine learning and others.

Jupyter is a project that is included in the Anaconda distribution, and is a browser-based application that allows the creation of Jupyter Notebooks, which can contain live code, documentation, interactive visualizations, etc. It's format is similar to iPython, because it is a project that was developed from iPython. You can save Jupyter Notebooks and share them with others. You can run Jupyter by itself or run it from the Anaconda Navigator (recommended). Here is a little YouTube tutorial that shows you how to set up and use Jupyter Notebooks.

Here's a screenshot of a start of a Jupyter notebook. In cell 3, I clicked on the "run" button to execute the Python there, which printed out the doubled numbers.


jupyter
Untitled2 Last Checkpoint: a minute ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Code
In [1]:
numbers = [3, 6, 23, 17]
In [2]:
def print_doubles(nums):
for num in nums:
print(f'{num} doubled is {2 * num}')
In [3]:
print_doubles(numbers)
3 doubled is 6
6 doubled is 12
23 doubled is 46
17 doubled is 34
In [ ]:

Have fun!