emacs/xemacs
python version

DR. YIHSIANG LIOW   (SEPTEMBER 21, 2025)

Contents

1 What is emacs/xemacs?

Emacs is a class of feature-rich text editors, usually characterized by their extensibility. Emacs has, perhaps, more editing commands compared to other editors, numbering over 1,000 commands. It also allows the user to combine these commands into macros to automate work.

Development began in the mid-70s and continues actively as of 2009. Emacs text editors are most popular with technically proficient computer users and computer programmers. The most popular version of Emacs is GNU Emacs, a part of the GNU project, which is commonly referred to simply as "Emacs". The GNU Emacs manual describes it as "the extensible, customizable, self-documenting, real-time display editor." It is also the most ported of the implementations of Emacs. As of July 2009, the latest stable release of GNU Emacs is version 23.1.

Aside from GNU Emacs, another version of Emacs in common use, XEmacs, forked from GNU Emacs in 1991. XEmacs has remained mostly compatible and continues to use the same extension language, Emacs Lisp, as GNU Emacs. Large parts of GNU Emacs and XEmacs are written in Emacs Lisp, so the extensibility of Emacs' features is deep.

The original EMACS consisted of a set of Editor MACroS for the TECO editor. It was written in 1976 by Richard Stallman, initially together with Guy L. Steele, Jr. It was inspired by the ideas of TECMAC and TMACS, a pair of TECO-macro editors written by Steele, Dave Moon, Richard Greenblatt, Charles Frankston, and others.

In Unix culture, Emacs became one of the two main contenders in the traditional editor wars, the other being vi. The word "emacs" is often pluralized as emacsen, by analogy with boxen (itself used by analogy with oxen) and VAXen.

– from wikipedia.org

XEmacs is a graphical- and console-based text editor which runs on almost any Unix-like operating system as well as Microsoft Windows. XEmacs is a fork, developed based on a version of GNU Emacs from the late 1980s. Any user can download, use, and modify XEmacs as free software available under the GNU General Public License version 2 or any later version.

– from wikipedia.org

For this tutorial, I assume you have access to Unix. So one of the following is true:

- You have a user account at `gandalf.ccis.edu` or `bilbo.ccis.edu`. You have read the PuTTY tutorial and know how to use PuTTY (or some other SSH client) to login to your account at `gandalf.ccis.edu` or `bilbo.ccis.edu`.
- You have access to a Linux machine (real or virtual).
- You have access to Cygwin on a Windows machine (this is not really Unix but a Unix emulation.)

Everything you learn in this tutorial applies to any Unix system with emacs/xemacs installed.

Emacs and xemacs are two very powerful text editor. I will only talk about some of the most commonly used operations.

Many of the operations uses special key bindings. These are special commands that you issue to emacs or xemacs from the keyboard. In fact you don't have to use the mouse at all. When you see this:

```
C-x-s
```

or

```
C-x C-s
```

it means "hold the [ctrl] key down, press and release `x`, and press and release `s`" (after that you may release the [ctrl] key). That's a command that you issue to save your file in emacs/xemacs. If you see something this

```
M-g
```

It means "hold the [alt] key and press and release the `g` key" (after that you may release the [alt] key.) You can also use the [esc] key. A quick warning: if you see something like
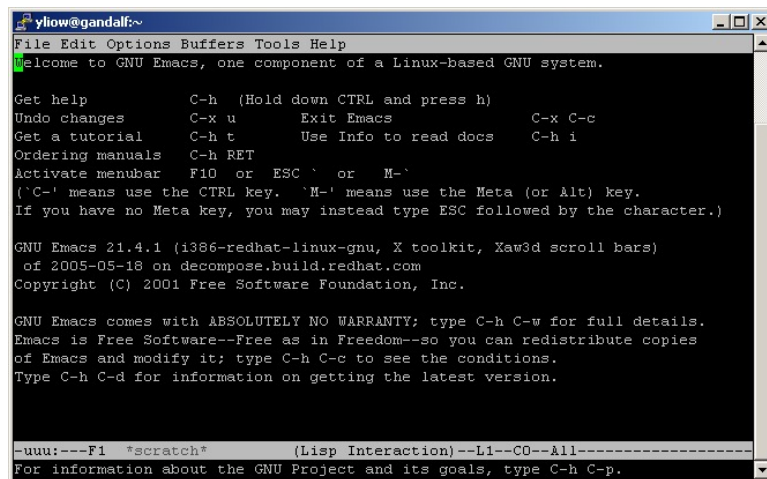
```
C-x 1
```

it means do "`C-x`" and then press `1`. So you do not hold the [ctrl] key down when you press `1`.
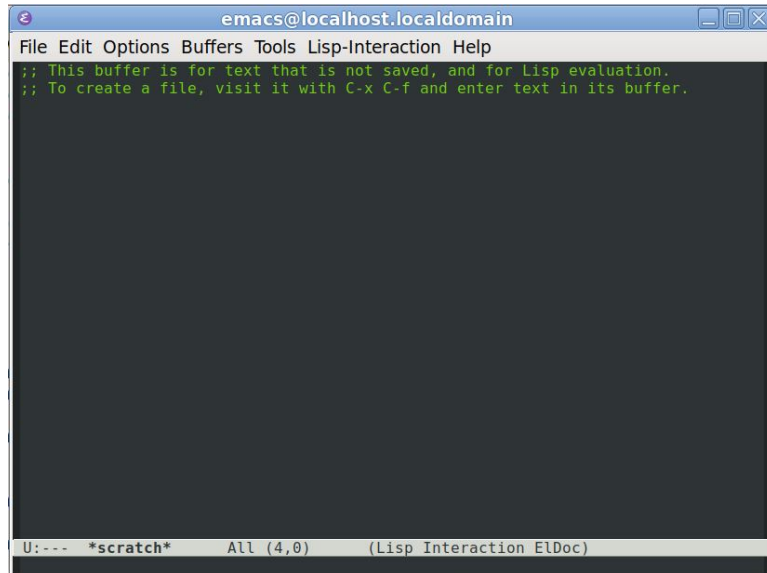
2 Starting and exiting xemacs

To run emacs, go ahead and type this into your shell:

```
[student@localhost ~]$ emacs
```

You will see this:

or

(What you see as the initial/welcome screen depends on the version and the configuration.) This runs the emacs program. To run xemacs, do this instead:

```
[student@localhost ~]$ xemacs
```

Emacs and xemacs are very similar.
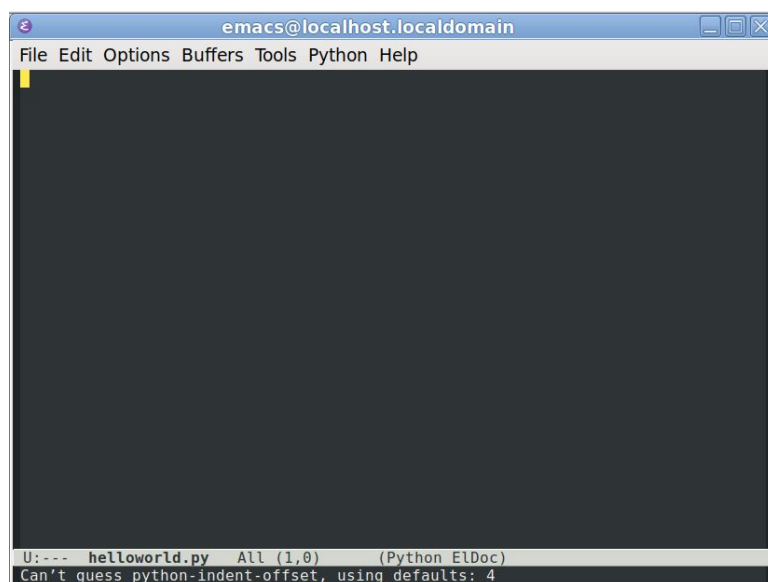
To exit emacs, in emacs do

```
C-x-c
```

This means "hold the [ctrl] key down, press and release the x key, and press and release the c key". You can release the [ctrl] key after that.

Please use one hand (your left) to do that!

Exercise 2.1. Start and exit emacs ten times. You have 10 seconds. ☐

Another way to start emacs is to open a file while you start the program. Try this in your bash shell:

```
[student@localhost ~]$ emacs helloworld.py
```


This will start emacs by loading the file `helloworld.py`. If `helloworld.py` exists, that file is loaded; otherwise a new file, `helloworld.py`, is created.

If you are using linux with GUI you should do

```
[student@localhost ~]$ emacs helloworld.py &
```

to run emacs in the background. This will make sure that your bash shell still respond to you.

3 File operations

Type this into your `helloworld.py`:

```
print("hello world")
```


To save the file do this:

```
C-x-s
```

Now exit emacs (recall: `C-x-c`) and you're back to your shell. At your shell's prompt view the contents of `helloworld.py` to verify that it's saved correctly execute the following in your shell:

```
[student@localhost ~]$ less helloworld.py
```

The program `less` will show you the contents of your file in your bash shell. To stop `less`, enter `q`.

To run your `helloworld.py`, do

```
[student@localhost ~]$ python helloworld.py
```

and you'll see `hello world` appear in your bash shell:

```
[student@localhost ~]$ python helloworld.py
hello world
[student@localhost ~]$
```


Exercise 3.1. Remove `helloworld.py` (use `rm`). Recreate the file all over again, save and exit, remove the file. Do this five times altogether. You have 2 minutes. □

Exercise 3.2. Take a simple assignment from CISS145 (or just make up one yourself) and do the above activity five times or until you can remember how to start emacs, save a file, and exit emacs. □

If you want to open another file while you're already in emacs, you do

```
C-x-f
```

At this point emacs will prompt you for the filename. In fact you can specify the full path and enter the filename and press enter. Using the [tab] key emacs will perform path completion for you. You can also enter . (a period) at the bottom and emacs will give you a directory listing from which you can select a file to be loaded.

Exercise 3.3. Open emacs by loading a new file `blah.txt`. While in emacs, load `helloworld.py`. Exit emacs. □

If you want to insert a file into the file you're working at the point of your cursor you do this:

```
C-x-i
```

Exercise 3.4. Load emacs, creating a new file `helloworld2.py`. Type some text. Now insert `helloworld.py` into `helloworld2.py` at any point in your text. Save your `helloworld2.py` and exit emacs. View the contents of `helloworld2.py` using `less`. □

4 Editing LaTeX files (added 2024-09-17)

When you are editing LaTeX files (files ending with `.tex`), to get a double-quote, you need to press the double-quote key twice.

5 Clearing a command

Before we go further, I should let you know that sometimes you might be in the middle of a sequence of commands and you want to cancel the commands. The easiest way to do that is to press the [esc] key several time.

Let's try this. Recall that `C-x-f` load a file. Suppose you do this:

```
C-x-f
```

The bottom status bar of emacs now ask you for a filename. At this point, press the [esc] key a couple of times and you will see Quit. This will end the command to load a file.

Easy right?

6 Moving around a document

Besides moving around the document with arrow keys, page up and down, there are several other things that you will find useful.

For a programmer, one of the most common action is to go to a particular line based on its line number. Open emacs (in the background) with your `helloworld.py` and change it to this:

```
print("hello world")
print("hello world"
print("hello world")
```

(i.e. remove the close parenthesis for the second statement.) Save your file. Run the program in your shell with

```
[student@localhost ~]$ python helloworld.py
```

and you should get an error:

```
  File "helloworld.py", line 3
    print("hello world")
        ^
SyntaxError: invalid syntax
```

This tells you that the python interpreter encounters an error at line 3. So now you want to open your program, go to line 3, and see if you can spot an error before the `print`.

To go to line 3 by doing this:

```
M-g-g
```

(i.e. hold the [alt] key, press and release g twice) you will be prompted for a line number. Enter 3 (and pressing the [enter] key) and your cursor will go to line 3.

Get it?

Exercise 6.1. Correct the error in `helloworld.py`, save the file, exit emacs, run your program to verify that the error is corrected. Repeat the whole process (creating an error somewhere, compiling to get the line number close to the error, go to the line, correct the error, save the file, exit emacs, compile the program) 5 times. □

Instead of moving one character at a time using the arrow keys, you can also move word-wise. To move right by one word you do this:

```
C-[rightarrow]
```

To move left by one word do this:

```
C-[leftarrow]
```

If you have a long line you might want to move quickly to the end of the line. Do this:

```
C-e
```

(or use the [end] key) and to move to the front of the line you do this:

```
C-a
```

(or use the [home] key.)

Exercise 6.2. Open your `helloworld.py` and practice the following: Move to line 2, move right by one word, move to the beginning of the line, move to the end of line Now make up a few more of such exercises.

If your document is huge it might be convenient to also know how to get to the top and the bottom of the document. To go to the top do

```
M-<
```

and do this is you want to go to the bottom:

```
M->
```

Exercise 6.3. Obviously, you should now create some huge text documents and practice the above two commands.

7 Undo

Probably the most important command is undo. Open a file in emacs, type in some random data and then do this:

```
C-/
```

or

```
C-x u
```

or

```
C-_
```

I usually do `C-/`.

8 Search and replace

Make a copy of your `helloworld.py` and call the new file `helloworld1.py`. Open `helloworld1.py` with emacs. Modify it to get this:

```
print("hello world")
print("hello galaxy")
print("hello columbian")
```

Let's do a search for the word `print`. To forward search for a string you do this:

```
C-s
```

followed by the string you are searching for. To search forward to the next occurrence of the string you entered, you continually do `C-s`. Go ahead and do it.

To search backwards (reverse search), you pretty much do the same thing except that you use

```
C-r
```

instead of `C-s`. Go ahead and search backwards for `print`.

Go to the line 1 of your file. Let's make the program say good-byes. We need to replace the `hello` with `good-bye`. First do this:

```
M-%
```

(You need the shift key to get to %.) emacs will prompt you for the string to replace. Enter

```
hello
```

and then enter the string you want to replace `hello` with. Enter

```
good-bye
```

emacs will find each occurrence of `hello` and ask you if you want to replace it with `good-bye` . You enter `y` (for yes) or `n` (for no) until emacs can't find anymore matches for `hello`. Note that emacs search forward. Go ahead and change your program to this:

```
print("good-bye world")
print("good-bye galaxy")
print("good-bye columbian")
```

Exercise 8.1. Using emacs replacement, get your program to look like this:

```
print("good-bye world", end="!\n")
print("good-bye galaxy", end="!\n")
print("good-bye columbian", end="!\n")
```

Save and exit. In your shell run your program:

```
[student@localhost ~]$ python helloworld.py
good-bye world!
good-bye galaxy!
good-bye columbia!
```

Exercise 8.2. Open `helloworld.py` using emacs. Using two search and replace modify the file

```
print("hello world")
print("hello galaxy")
print("hello columbian")
```

so that it looks like this:

```
print("hello world\n" +\
      "hello galaxy\n" +\
      "hello columbia")
```

Save and exit. In your shell run your program to get this output:

```
hello world
hello galaxy
hello columbia
```

☐

Exercise 8.3. Redo this whole section 3 times.   ☐

9 Selecting, copying, pasting, and deleting a region

To copy-and-paste a chunk of text, first you need to select the region to copy. You also need to do that if you want to delete a region. So let me first show you how to select a region ...

Selecting a Region. There are two ways to do this. First go to one end of the region. Now hold the shift key down and then use the arrow keys or pageup/pagedown key to go to the other end. You will see that the text you go over is highlighted – you're basically selecting a region.

The above method does not work if you're in a pure text-based environment for instance if you use some ssh program to login to your secret unix account in Russia. So another way to do this is to go to one end of the text you want to select and do

```
C-spacebar
```

and then go to the other end. In this type of environment, you might not see the highlighting.

Of course you can use your mouse to select a region too.

Copying. At the other end of the region you want to copy, do

```
M-w
```

Now emacs has a copy of the region that was marked.

Paste a Region. Now go to the place where you want to paste the region and do

```
C-y
```

This is called "yank" it in.

Delete a Region. To delete a region, again you have to select the region first. Once your region is selected, do

```
C-w
```

This is called "wipe" it out.

Exercise 9.1. Write a helloworld program:

```
print("hello world")
```

Now using copy-and-paste to do this:

```
print("hello world")
print("hello world")
print("hello world")
print("hello world")
```

Next change your program to this by removing a region:

```
print("hello world")
print("hello world")
```

Redo this exercise 5 times.   □

Exercise 9.2. First write this python program:

```
x = int(input())
print(x)
```

Change the above to this using select and copy:

```
x = int(input())
print(x)
x = int(input())
print(x)
x = int(input())
print(x)
```

Change it to this using replace:

```
year = int(input())
print(year)
x = int(input())
print(x)
x = int(input())
print(x)
```

Change it to this using replace:

```
year = int(input())
print(year)
month = int(input())
print(month)
x = int(input())
print(x)
```

FInally change it to this using replace:

```
year = int(input())
print(year)
month = int(input())
print(month)
day = int(input())
print(day)
```

10 Indentation

If your emacs is set up correctly you don't have to worry about indentation because emacs will do it for you automatically. There are however times when you need to re-indent. For instance, maybe you've accidentally inserted some extra spaces:

```
    print("hello world")
print("hello world")
print("hello world")
```

All you need to do is to go to first line and press the tab key. Your cursor need not be at the beginning of the text.

There are also times when you need to re-indent a whole region. For instance suppose you have this chunk of code:

```
if x == 0:
    if y == 0:
        x0 = 0
        x1 = 1
        x2 = 2
        ...
        x49 = 49
```

And then you realized ... duh! ... the code should be this:

```
if x == 0:
        x0 = 0
        x1 = 1
        x2 = 2
        ...
        x49 = 49
```

Now you need to indent all the 50 lines of code. You can use the tab key for each line to be indented. But the smarter thing to do is to indent a region instead of doing it one line at a time. So you first mark the region to indent (example: select the region using the shift key) and then do

```
C-M-\
```

That's it. Another way to do that is to do

```
M-x indent-region
```

You can use [tab] to complete emacs commands. Do this:

```
M-x inde[tab]-reg[tab]
```

Exercise 10.1. Write a dummy program like this:

```
x = 1
x = 1
x = 1
x = 1
x = 1
```

Now add this

```
if y == 0:
x = 1
x = 1
x = 1
x = 1
x = 1
```

and perform an indentation of a region to get it to look like this:

```
if y == 0:
    x = 1
    x = 1
    x = 1
    x = 1
    x = 1
```

Now make it go back to the original version. Do this 5 times. □

Instead of auto indent using [tab] of `indent-region`, manual indentation is sometimes necessary because there are times when emacs cannot possibly figure out what you want the code to do. For instance

```
if x == 0:
    y = 1
    z = 2
```

and

```
if x == 0:
    y = 1
z = 2
```

are both syntactically correct. Here's how you do manual indentation. To do forward indentation, you select the region and then you do

```
C-c >
```

To reduce indentation (called dedent), you do

```
C-c <
```

Exercise 10.2. First write this to a file:

```
if y == 0:
x = 1
x = 1
x = 1
x = 1
x = 1
```

Next manually forward indent every line except the first.

Exercise 10.3. First write this:

```
if x == 0:
    y = 1
    z = 2
    a = 3
    b = 4
    if c == 5:
        d = 6
        e = 7
        f = 8
```

Next, manually change the indentation to this

```
if x == 0:
    y = 1
    z = 2
    a = 3
    b = 4
if c == 5:
    d = 6
    e = 7
    f = 8
```

and then to this:

```
if x == 0:
    y = 1
    z = 2
a = 3
b = 4
if c == 5:
    d = 6
    e = 7
    f = 8
```

and then to this:

```
if x == 0:
    y = 1
    z = 2
    a = 3
    b = 4
    if c == 5:
        d = 6
        e = 7
        f = 8
```

and then this:

```
if x == 0:
    y = 1
    z = 2
    a = 3
    b = 4
    if c == 5:
        d = 6
e = 7
f = 8
```

11 Comments

When writing python source files, you frequently need to comment and uncomment code. Try this.

In your python source file, select a region and then do

```
M-;
```

You will see that the region is commented. If the region was already commented, then doing

```
M-;
```

will uncomment the region

Get it?

Exercise 11.1. First write this:

```
if x == 0:
a = 1
b = 2
c = 3
if y == 4:
d = 5
e = 6
f = 7
```

Then modify the above to get this in the most efficient way:

```
if x == 0:
    a = 1
#b = 2
#c = 3
if y == 4:
    d = 5
    #e = 6
    #f = 7
```

and then this:

```
if x == 0:
    a = 1
    b = 2
    c = 3
    #if y == 4:
    #    d = 5
    #    #e = 6
    #    #f = 7
```

and then

```
if x == 0:
    a = 1
    b = 2
    c = 3
    if y == 4:
        d = 5
        e = 6
        f = 7
```

12 Multiple frames

When you write a program, you will frequently work with multiple files. Of course you can run multiple instances of emacs, running one instance of emacss for each file. But in fact you don't have to because a single instance of emacs can manage multiple files, each file in a buffer in emacs.

In fact it's a very bad idea to open multiple files with multiple instances of emacs. Why? Because you might accidentally have one file opened in two separate instances of emacs and you might be making *different* modifications to the same file. Of course when you save the program in one emacs instance, earlier program changes saved using the other emacs instance will be lost!!! See the problem?

So let me show you how to run one instance of emacs to manage any number of files.

First run this to write a hello world program:

```
[student@localhost ~]$ emacs helloworld.py &
```

Write your `helloworld.py` but don't exit yet. While in emacs do

    C-x 2

You now have two frames: the original frame has split horizontally. Now do

    C-x o

(Note: No `C` for the `o`, i.e., you do `C-x`, release the Ctrl key and press `o`.) This will move your cursor to the "other" window. You can also use

    C-tab

Now do

    C-x 3

and you current frame will split vertically. Do

    C-x 3

a couple of times and you'll go from one frame to another. If you do

    C-x 1

and your current frame will fill up the whole window – all other frames disappear. Create a few more frames again.

Now let's load a different file into a frame. Go to any frame and do

    C-x-f

(see previous pages – I've already talked about `C-x-f` earlier.) At the bottom status bar, emacs will display your current path. You can enter a filename. If the file is not found, then emacs assumes you want create a new file. If the filename, emacs assumes you want to a load the existing file into the frame. In any case you now have two files, each file occupying one buffer.

If you want to switch to a new buffer, you can list all buffers by doing this:

    C-x-b

By the way, you can select a buffer by doing

    C-x b

(Note: No `C` for the `b`).

When working with multiple frames, usually you want to save changes to the files in all frames. You can of course all through all the frame and save the files one at a time and do `C-x-s`. But you can also save all the files in all the frames like this: do `C-x s` (no `C` for the `s`) and then when prompted, you enter `!`.

Exercise 12.1. First write this program using emacs:

```
print("hello world")
```

Close your emacs. How open the file in two separate emacs instances:

```
[student@localhost ~]$ emacs helloworld.py &
[student@localhost ~]$ emacs helloworld.py &
```

In one emacs, modify the program:

```
print("top")
print("hello world")
```

Look at the other emacs and you'll see:

```
print("hello world")
```

The second emacs does not see the change you made in the first emacs. Close both emacs instances without saving the change. Now open the original program again using

emacs. In emacs, split the frame vertically to get two frames. You'll see the same program in both frames. Modify the program in the left frame like this:

```
print("top")
print("hello world")
```

and you'll see the other frame catches the change. Modify the program in the right frame like this:

```
print("top")
print("hello world")
print("bottom")
```

and you'll see the other frame catches the change.

13 Running python in emacs/xemacs

Of course you can run your python source files in the shell. However you can run it inside emacs. You can also run a python shell inside emacs. In fact to run your python program you are writing, you have to first open a python shell.

The benefit of running a python program inside emacs, is that emacs can point you directly to the location of the program error.

Enough talk ... let's do something.

First run emacs and write this program:

```
print("hello world")
print(x)
```

Note the error.

While in emacs do

```
C-c-p
```

you will see a python shell inside emacs. You can test the python shell to make sure it works:

```
Python 3.7.9 (default, Aug 19 2020, 17:05:11)
[GCC 9.3.1 20200408 (Red Hat 9.3.1-2)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> python.el: native completion setup loaded
>>> print(1 + 1)
2
>>>
```

By the way, you can open the python shell in emacs without first loading a python program.

Now I'm going to run my program

```
print("hello world")
print(x)
```

in the python shell. First I make sure my cursor is in the frame with my python program. Then I do

```
C-c-c
```

I get this output in my python shell:

```
>>> hello world
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/student/helloworld.py", line 2, in <module>
    print(x)
NameError: name 'x' is not defined
>>>
```

If I put my cursor at the underlined text and press the enter key, emacs will point the cursor to line 2 of my program where the error is found. I make this correction

```
print("hello world")
print(x)
```

and do

    C-c-c

and get the following is my python shell:

```
Python 3.7.9 (default, Aug 19 2020, 17:05:11)
[GCC 9.3.1 20200408 (Red Hat 9.3.1-2)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> python.el: native completion setup loaded
>>> hello world
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/student/hw.py", line 2, in <module>
    print(x)
NameError: name 'x' is not defined
>>> hello world
0
>>>
```

Exercise 13.1. Write a simple python program in emacs that accepts two inputs from the user and prints the sum. Run that in emacs. □

You can actually run a selected region (instead of the whole program). To do that you select a region and then do

    C-c-r

14 Shell

In the previous section I talked about running your program inside emacs through a python shell. But it gets even better ...

In fact you can run a bash shell inside emacs and execute any linux command. So you can pretty much do everything inside emacs! To try it out, execute

```
M-x shell
```

and you will have a shell inside your emacs.

One thing to note: In your usual terminal bash shell, you can use the arrow keys to get the previous and next command in the bash history. However inside emacs when you're running a shell, the arrow keys move your cursor around in the shell output. So instead of using arrow keys, you use `M-p` for the previous shell command and `M-n` for the next shell command.

So now you know how to run your python program in a python shell inside emacs and inside a bash shell inside your emacs.

Exercise 14.1. Use emacs to write a python program. Make sure there are at least 3 errors in your program. Run your program within emacs (through a python shell or bash shell) and correct the errors one at a time, re-compiling after each correction. Run your program within emacs. Do the above 5 times.

15 What's next?

This is only a short intro to emacs. There are lots of emacs tutorials on the web. You can find out more using google. Another to do is to google for the "emacs refcard" (reference card) which contains a summary of emacs commands. Of course there's also the official emacs web site.

16 Summary

```
C-x-c            Exit emacs/xemacs
C-x-s            Save file
C-x-f            Load file in current frame
C-x i            Insert another file into current file


M-g M-g          Go to line number (M-g for xemacs)
C-[rightarrow]   Go to next word
C-[leftarrow]    Go to previous word
C-a              Go to beginning of line
C-e              Go to end of line
M-<              Go to top
M->              Go to bottom


C-/              Undo
C-x u            Undo
C-_              Undo


C-s              Search
C-r              Reverse search
M-%              Replace


C-spacebar       Mark beginning of region. (Or just hold the shift key.)
M-w              Copy region
C-y              Paste: "yank" it in
C-w              Delete region: "wipe" it out


[tab]            Indent a line
C-M-\            Indent region
C-c >            Indent region
C-c <            Dedent region


M-;              Comment/uncomment region


C-x 1            Keep 1 frame
C-x 2            Split frame vertically
C-x 3            Split frame horizontally
C-x o            Go to other frames
C-tab            Go to other frames


C-x-b            Show all buffers
```

```
C-x b          Select buffer

C-c-p          Open python shell
C-c-c          Run python program in python shell
C-c-r          Run region of python program in python shell
M-x shell      Shell
```

17 Summary: cursor movement

Without arrow keys:

```
                CHARACTER  WORD  LINE  PARAGRAPH  PAGE  FILE  LINE-NUM
right/forward   C-f        M-f   C-e   M-}        C-v   M-<   M-g M-g
left/backward   C-b        M-b   C-a   M-{        M-v   M->
up/previous     C-p
down/next       C-n
```

Using arrow keys:

```
                CHARACTER  WORD
right/forward   RIGHT      C-RIGHT or M-RIGHT
left/backward   LEFT       C-LEFT or M-LEFT
up/previous     UP
down/next       DOWN
```