

CS279 Python and Terminal Tutorial (2023)

Ruhi Sayana



Outline

- **Terminal**
 - Basic terminal commands and tips
 - Editing and running python scripts
 - Demo
- **Python**
 - Variable types and operations
 - Lists, tuples, and numpy arrays
 - If-else, loops, and functions

Terminal

Basic Commands

General syntax: **command** [-flags] <arguments>

- ls <destination (default: current)>: list out the contents of a directory
 - ls -lh: lists out contents of directory and their file sizes
- pwd: lists file path for current directory (from root)
- cd <destination (default: home)>: navigate through directories
 - cd assn1
 - cd .. to go to parent directory
 - cd ~ to go to root directory
- mv <source_file> <destination>: move file to a new directory
 - mv <source_file> <new_file>: renames source_file with name in new_file (can be used for directories also)
- cp <source_file> <destination>
- rm my_file: deletes file (**cannot be recovered: use cautiously**)
 - rm -rf my_dir: deletes my_dir and all its contents (including subfolders)

Tips

- Use tab to auto-fill file names
- Use up/down arrows to go through previous and new commands
- **clear**: clear the terminal
- **head/tail <filename>**: print first few or last few lines
- **man <command>**: more information about a command (q to exit)

Editing files and running python scripts

- **Editing files**
 - **Recommended:** using an IDE (PyCharm, VSCode)
 - VSCode download: <https://code.visualstudio.com/Download>
 - PyCharm is already installed on LTS machines
 - **Command Line Editors**
 - nano <filename>
 - Creates a new filename if <filename> does not exist
 - Ctrl+X to exit, y to save, Enter to confirm
 - Other options: emacs or vim
- **Running Python scripts (see Software handout)**
 - python hello_world.py <any arguments>
 - hello_world.py is the filename
 - LTS users: python3.9 instead of python
- Installing packages with pip
 - pip install <package> (for example, pandas)

Questions

Python

Variables and Operations

- Declare a variable: `var = value`
- Types
 - Integer: -10, 0, 5
 - Float: -5.4, 1.3, 10.0
 - String: "strawberry", 'i enjoy country music'
- Casting: `int()`, `float()`, `str()`
 - `int("3") = 3`
 - `str(9.32) = "9.32"`
 - `float(2) = 2.0`
- Operations: +, -, * (multiplication), / (division), ** (exponentiation)

Data Structures: Lists

- `my_list = [-1,3,3,4,5,6]`
 - Empty: `my_list = []`
- Lists are ordered and mutable
- Indexing
 - Lists are zero-indexed: `my_list[0] = -1`, `my_list[len(my_list) - 1] = 6`
 - Negative indexing: `my_list[-1] = my_list[len(my_list) - 1] = 6`
 - Slicing: `my_list[start : stop : step]` (default: `start = 0`, `stop = len(my_list)`, `step = 1`)
 - `my_list[2:4] = [3, 4]`
- Adding elements: [`my_list.append\(element\)`](#), [`my_list.insert\(index, element\)`](#)
- Removing elements: [`my_list.remove\(element\)`](#), [`my_list.pop\(index\)`](#)
- Other useful functions: [`sum\(my_list\)`](#), [`len\(my_list\)`](#), [`max\(my_list\)`](#), [`min\(my_list\)`](#)

Data Structures: Tuples

- Declaring a tuple: `my_tuple = (1,2,3)`
 - `my_var = 1, 2, 3` will generate a tuple by default
- Ordered, immutable data structure (tuples cannot be changed once created)
- Indexing and slicing identical to lists
- Common aggregate functions: [`sum\(my_tuple\)`](#), [`len\(my_tuple\)`](#), [`max\(my_tuple\)`](#), [`min\(my_tuple\)`](#)

Data Structures: numpy Arrays

- pip install numpy (should already be installed if you use anaconda for python)
- import numpy as np
- Initialization
 - From python lists: `array_1d = np.array([1,2,3,4])`, `arr = np.array([[0,0,0],[0,0,0]])`
 - To a single value: `np.zeros` or `np.ones`
 - `zero_arr = np.zeros((2,3))`, `one_arr = np.ones((4,2))`
- Can do matrix operations and fast operations
- Indexing: `my_arr[row_index(s), col_index(s)]`: `my_arr[1:3,]`
- Element-wise comparisons:
 - `a = np.array([1,2,3])`, `b = np.array([2,3,4])`
 - `a > b`
 - `array([False, True, True])`
- Can do element-wise arithmetic as well and vectorized operations

Conditionals, Loops, Functions

If-else statements

- Boolean keywords: True and False
- Comparators
 - == equals
 - != not equals
 - < less than
 - > greater than
 - <= less than or equal to
 - >= greater than or equal to
- Combining Boolean values: *and*, *or*, *not*
- Membership: *in*
- Careful with indentation!

Loops

- For Loop: loop for a fixed number of times
 - `for i in range(4):` *#can also do range with (start, stop) or (start, stop, step_size)*
 `do_func()`
 - `for elem in list:` *#for-each loop*
 `do_func(elem)`
- List comprehension (related)
 - `my_list = [val for i in range(4)]` → outputs [0,1,2,3]
- While loop
 - Loop many times while condition is true
 - `while (j > 4):`
 `action()`
 `j = j - 1`
- Careful with indentation!

Loops

- For Loop: loop for a fixed number of times
 - `for i in range(4):` *#can also do range with (start, stop) or (start, stop, step_size)*
 `do_func()`
 - `for elem in list:` *#for-each loop*
 `do_func(elem)`
- List comprehension (related)
 - `my_list = [val for i in range(4)]` → outputs [0,1,2,3]
- While loop
 - Loop many times while condition is true
 - `while (j > 4):`
 `action()`
 `j = j - 1`
- Careful with indentation!

Functions

- `def function_name(param1, param2):`
 `return param1 * param2 / (param1 - param2)`

- Can return multiple values as a tuple

```
def func(param1, param2):  
    return param1, param2  
result = func(1,2) # result is (1,2)  
a, b = func(1,2) # a = 1, b = 2  
a, b, c = func(1,2) # throws error
```

- Can return lists, booleans, numpy arrays, or any type – do not need to specify
- **Careful with indentation!**

Questions

Resources

- [Detailed python review from CS229](#): This has much more information than what is needed for the scope of this class, but some useful sections on plotting, more data structures, and in-depth numpy/list examples
- [Tutorial for using python with VSCode](#)
- [Python](#) and [numpy](#) tutorials