**Shell**

* Simple Command

Open a shell: e.g bash

Exit a shell: exit()

Man whoami ##not work on Windows

Sleep 10 ## 10 second

When you make mistake and want to go on, hit Ctrl+c

Clear# clear the screen

* Files

Root directory-basic folder on machine

>pwd

C/Users/Administrator

>ls

List the folder and files names in

>tree

(only works on Mac)

>ls –F

>ls –a

>cd Doc(Tab)

Returning all files name start with Doc

>cd or cd~

Return to the main root folder

>history

Return all the orders written

>! # shown in the *history* results

>mkdir #make a new folder

>cd a\ bad\ name

>nano (create a file only in Mac)

>touch refs.txt draft.txt (return empty file)

>cat (reading files)

>rm draft.txt #remove the files

>rmdir # remove directory. If there are files in the directory, you cannot remove it. So you need to delete the files first.

>rm –r /Users/ivan/my\_paper# delete everything in the directory!!!

>rm\* #remove anything

>ls \*.dat (return anything end with .dat)

>ls\*e\* (return files have a e in their names)

>mv #rename folder or file

**Python**

>ipython #(enter python)

>print(“Hello”)

>%run hello.py #run a file in python

No type classification in Python.

>can class a variable using type()

If we divide two integer, Python will give a integer.

>2/5

0

>2/5.0

0.4

>2/float(5)

0.4

>raw\_input()

>message=”Hello”+”Names” ## ‘,’ only works in ‘print’

>age=32

>message2=”You age will be”+str(age)

>print “abc”\*4

>name.lower()

Python start with zero

1. list []

>a=[1,3.4,”abc”

>a[0] returns 1 ##left to right start with 0

>a[-2]returns 3.4 ##right to left start with -1

>a[1:3] returns [3.4,”abc”] ## except the element in the third place

>a=[1,2,3,4]

>a.append(5) ##add a new element

>a.extend([6,7]) ##

>a.remove(2)

>del a[2]

>sorted(c) ##not change what is in c

But if we use >c.sort() c will change

>set() ## return the a value in certain list

Dictionaries e.g. e{} there is no order in a dictionary

|  |  |
| --- | --- |
| Key | Value |
| 1 | “Al” |
| 2 | “Betty” |
| 3 | “Cast” |
| 4 | “Dave” |
| 5 | “Esme ” |
|  |  |

>e.keys() returns [1,2,3,4,5]

>e.values()

>e.items() #return both the keys and the values

>del e[1]

>1 in e returns Ture

>6 in e returns False

And if you want to add a new key in a dictionary, >e[5]=”Cety”

2. tuple ()

Like list, but once you create a tuple, you cannot change the value or length or anything in it.

Change a tupo to list

>c=list(b)

3.loop

>i=2

>j=3

>i==j #doing comparing using double = similarly != means not equals

Price=3.4

Quantity=5

Buy=False

Fruits=[]

If not (price<3.5):

If quantity>=5

Print “BUY” (buy=True fruits.append(“apple”))

Else:

Print “SELL” (buy=False)

If buy:

Print “BUY”

Else:

Print “SELL”

Price=3.4

Quantity=4

Fruits=[]

If price<3.5:

If quantity>=5:

Fruits.append(“apple”)

Elseif: price<4.0

Fruits.append(“banana”)

Else:

Fruits.append(“banana”)

Course Material:

* <http://bit.ly/16wM72A>

=======

The Shell

=======

- opening the shell

-- Mac: open Terminal Application

-- Windows: open Git Bash Application

- why?

-- today use mouse, graphical environment or touch and gestures

-- before, all commands were textual instructions to the computer

- TRY IT: Type "whoami" and press Enter

- shell helps to automate mundane tasks

- disadvantages: cryptic

- there are different types of shells just like there are different word processors

-- we are using bash

- running a shell

- TRY IT: Type "bash" and press Enter, Type "exit" to close the "bash" program

-- the text preceding the cursor is called the prompt

- Commands:

-> whoami

-> echo: TRY IT: echo "hi guys"

--- it should print it back to you

-> man: TRY IT: man whoami

--- it is the manual page for whoami

--- to quit the "man" application, hit the "q" key

-> clear: just clears the screen

-> sleep: sends the shell to do nothing for a number of seconds ("sleep 10")

-- Ctrl+C will end a program

-> if you were to type "sleep 1000000", it would take a very long time, Ctrl+C will quit it

- shell programs are very small, do a single thing, programs like Word can do a lot of things

- can put together lots of shell programs to build larger program

Directories Example:

Navigating the file system in the shell (just like opening/closing folders in the Mac Finder or Windows Explorer)

will be using a set of directories (aka folders): thesis and then a few subdirectories (biblio, chaps, figs),

/ indicates a "branch", what ever is below is a file or subfolder

thesis is the parent folder of biblio.  biblio is the child folder of thesis

the "root branch" or root directory is just the forward slash /

-- the "path" is the list of all folders with forward slashes separating things

-- it is the address of a particular folder or directory

-> pwd: shows the current directory

-> ls: shows the contents of the current directory

-- shell allows you to automate things (may seem slower to find a single file)

-> TRY IT: "ls -F" shows differences between files and directories

-> "ls -a" shows "hidden files", files only the computer uses

-- dash (minus) plus a character is a an \*option\* to a shell program

-> TRYIT: "ls -a -F" combines two options

-> new command: "cd" is "change directory"

-> TRY IT "cd /", "cd ~"

Tree command doesn't work on my Terminal window (OSX); typing: **find . -print | sed -e 's;[^/]\*/;|\_\_\_\_;g;s;\_\_\_\_|; |;g' > file helps**

cd without any argument goes to the home directory

-- if you get lost, cd takes you back home

-- "cd ~" is the same thing

shell keeps track of history!

- it shows past commands with a number as a prefix

- can use the number to run the command again

- type exclamation mark (!) plus the number (e.g !835)

- also up and down arrows will let you go up and down your history, just keep pressing up arrow until you see the command you want to run again, then hit enter

-- !! Auto-completion !! -- hit the TAB key to complete names (works well for long names, type the first characters and let TAB do the rest of the work)

Commands

-- mkdir: make a new directory

-- touch: creates a file

-- cat: prints the contents of a file

-- rm: remove a file

Creating folders and files (example):

> cd

> ls

> mkdir my\_paper

> lsl

## you should see a new folder named my\_paper in the list

> mkdir "a bad name"

## you could also do > mkdir a\ bad\ name

## \ escapes the next character so it isn't treated as a separator

> cd a bad name

## this will \*\*not work\*\*, gives an error

## suggestion: do not use whitespaces when naming files and directories

> nano draft.txt

## nano may not be installed on Windows

## add some text and save it

> touch refs.txt

## create an empty file on Windows named 'refs.txt'

> cat draft.txt

> cat refs.txt

## You can also look at these directories and files in the normal GUI interfaces

## You can also edit the files in a program like Notepad and then "cat" it again in the shell

## CAREFUL with the next command, there is no way to recover the file

> rm refs.txt

## Deleting directories

> rm "a bad name"

## It doesn't work, error about it being a directory

> rmdir "a bad name"

> ls

## it is gone from the directory listing!

## now do something bad-- delete my\_paper

> rmdir my\_paper

## Error: the directory has a file in it

> cd my\_paper

> rm draft.txt

> ls

## nothing here now

> cd ..

> rmdir my\_paper

## now it works (assuming there are no files left)

## new command

> mkdir my\_paper

> cd my\_paper

> touch draft.txt refs.txt biblio.txt notes.txt

> ls

> rm drafts.txt notes.txt

## another shortuct - star/asterisk

## \* means everything in the directory!

> rm \*

## removes the rest of the files

> ls

## no files left

> touch draft.txt refs.txt biblio.txt notes.txt

> touch a.dat b.dat c.dat

## see only the data files

> ls \*.dat

## see all the files with an "e" in their names

> ls \*e\*

## \* is also called a "wildcard"

> touch 2013-10-06.txt 2013-10-14.txt 2013-11-02.txt

## delete only the files from october

> rm 2013-10-\*

-- new command: rename files and directories

> ls \*.dat

## run ls first to check what you might be removing

> rm \*.dat

> ls

## change name from draft.txt to final\_version.txt

> mv draft.txt final\_version.txt

> ls

## file should have a new name now

## file's contents are exactly the same

## can also rename directories

> cd ..

> mv my\_paper my\_thesis

## copy the final version

## cp is the copy command

> cd my\_thesis

> mkdir send\_to\_advisor

> cp final\_version.txt send\_to\_advisor/my\_thesis.txt

Q: What if there are a thousand files in the folder?

A: Can use rm -r, but this is a scary command, it can delete lots of stuff

Q: Does typing "clear" clear your history as well?

A: Try it :), but no it does not, it just clears the screen

Q: Can you create a file the same way using Sublime?

A: Yes, Notepad or Sublime or any other editor can create or edit files

=====

Python

======

(Sorry, I've deleted the previous text)

Hello, world program in Python

--------------------------------------------

1. Open an editor, write this: print "Hello world"

2- Save it to a file called hello.py

3. Run it:

 - from the terminal/shell: ipython hello.py

 - from ipython itself:

     a) open ipython from the terminal: ipython

     b) do: %run hello.py

Variables and types

-----------------------------

Assigning a variable means creating a "thing" and making it equal to "something":

 E.g. a = 1.2, creates "a" and makes it equal to 1.2

 The same variable can be assigned to anything else later in the program:

 a = "NYC rocks"

 Types are the different kinds of "things" that you can have inside your program. These include integers, floating point number, strings, lists, and even more complex objects that you can define.

 "Things" are called objects more properly. So we say string are a type of object, or integer are another type of object. Objects have methods that make them functional. You get a method by using "." after the variable name. E.g.:

 a = "I'm a string"

 a.upper() will create a new string capitalized

 To get a list of the methods of an object use help:

 help(string)

 Objects, variables, and all that

 --------------------------------------------

 Everything in Python is an object: a string is an object, a list is an object, a function is an object, etc...

 A variable is an object also, but it can make equal to any other object! Once you assign a variable it takes the type of the object you assigned to, until you reassign it to something else.

 You can extend, slice, append,... lists. You can even mix up different types of objects in the same list.

 Lists vs tuples

 ---------------------

 You create a list with square brakets, but you create a tuple with round brakets

 The difference is that tuples are inmutable, i.e. you cannot change them once you assigned them

 Use case: use list for a list of people in the class (becuase it can change, new people join...), but use a tuple with 3 elements for the names of the people (they will have a first name, middle and last name) but their names are not gonna change once assigned, nor they are gonna get extra names later.

 Aliasing

 -----------

 Aliasing is a bit confussing, and error-prone at first. It means that you give several names (i.e. variables) to the same object (or value)

 a = [1, 2]

 b = a

 a.append(3)

 Now a = [1,2,3], but also b = [1,2,3]

 Containers

 ---------------

 Lists and tuples are examples of containers, with are just collections of objects. There are a bunch of them in Python: sets, dictionaries, etc...

 All work pretty much the same way, but have different methods and properties to do slightly different things.

 Dictionaries

 -----------------

 Dictionaries are containers with pairs of keys and values.

 Values can be anything, but keys must be inmutable, i.e. they cannot change once assigned: you can use tuples, but not lists.

Exercises:

# 1. Write a script that asks for a user's first name and last name

# and prints Hello <first name> <last name>

# Example: Enter your first name: David

# Enter your last name: Koop

# Hello David Koop

# Solution

first = raw\_input("Enter your first name: ")

last = raw\_input("Enter your last name: ")

print "Hello", first, last

# 2. Ask for a user's age, then print what their age will be next year

#

# Enter your age: 34

# You will be 35 next year

# Solution

age = raw\_input("Enter your age: ")

print "You will be", int(age)+1, "next year"

# 3.  Given a dictionary that shows the number of people in a given zip code 36082, print the most number of people in a zip code?

# the dictionary is below

# dictionary is zip code as key, # of people is value

zip\_population = {13441: 4581, 31362: 4183, 68612: 2274, 16391: 8786, 93833: 8472, 14730: 436, 32271: 2260, 70131: 3895, 70801: 2519, 17109: 5344, 15171: 8229, 91156: 32, 31509: 7742, 73238: 934, 81156: 903, 24476: 862, 22174: 6352, 28066: 9023, 25253: 7903, 56104: 9649, 93866: 2019, 31150: 6486, 38216: 9713, 24882: 6036, 59827: 7314, 69558: 4350, 30779: 4371, 73538: 325, 15555: 5211, 15814: 9762, 63688: 1450, 87882: 1432, 28624: 6252, 67795: 1189, 29012: 3334, 49365: 2490, 13399: 5454, 89075: 9685, 46432: 6054, 56684: 4598, 40304: 6531, 24561: 7191, 36082: 4864, 68243: 4224, 80501: 7536, 16248: 3695, 46931: 3702, 95869: 8534, 92926: 8369, 57301: 3304}

Answers:

Nil Simsek:

In [151]: a = zip\_population.values()

In [152]: b=sorted(a)

In [153]: b[0]

Out[153]: 32

In [154]: b[-1]

Out[154]: 9762

In [155]: print "Maximum is", b[-1]

Maximum is 9762

Felicia:

b=zip\_population.values()

c=sorted(b)

print "Maximum number of people in any zip code:", c[-1]

Sabrina:

max(zip\_population.values())

p= sorted(zip\_population.values())

len(p)

p[49]

Gabi:

val\_list = zip\_population.values()

print val\_list[-1]

Erin:

d=sorted(zip.population.values())

print"Maximum pop:", d[len(zip\_population)-1]

Booleans and testing for true or false

------------------------------------------------------

Python has a boolean type, therefore you can have variables of boolean type. The boolean types are True and False

To deal with boolean types you have boolean operators: ==, <, >, etc...

The operator to negate is !

a = 1

b = 2

a == b (False)

a < b (True)

a != b (True)

Conditionals

------------------

In programming you use conditional to control the "flow" of your program, i.e. the program can go one way or another other. Conditional are created using "ifs".

 if price < 3.5:

    print "buy"

else:

    print "sell"

Be very careful with the indentation. This is the most common error for beginners. Indent with 4 spaces.

"ifs" can be nested, so you make your program to go whenever you need to.

"elif" is just a shorcut for else if and you use it when your conditional has more than two branches. Your last branch always uses "else".

Loops

---------

Loops are another way to control the "flow" of your program. They are use to repeat a block of code mutiple times.

You create loops with "while" and "for". Loops run until a condition is met.

Example with while:

count = 0

while count < 10:

    print count

    count = count +1

will print 0, 1, ...., 9

Example with for:

fruits = ["apple", "orange", "banana"]

for fruit in fruits:

    print "Buy "+fruit

will print apple, orange, banana

For-loops are the perfect way to run thru (called iterate) containers like lists, tuples, sets, and dictionaries.zip = zip\_population.keys()

population = zip\_population.values()

max\_value = max(zip\_population.values())

for (zip,population) in zip\_population.iteritems():

        if population == max\_value:

                print "Maximum population at", zip, "is", population

Exercise 4:

Print the zipcode with the maximal population

Using the new zip\_population dictionary, print all zip codes with the maximal population

zip\_population = {13441: 4581, 31362: 4183, 68612: 2274, 16391: 8786, 93833: 8472, 14730: 436, 32271: 2260, 70131: 3895, 70801: 2519, 17109: 5344, 15171: 8229, 91156: 32, 31509: 7742, 73238: 934, 81156: 903, 24476: 862, 22174: 6352, 28066: 9023, 25253: 7903, 56104: 9649, 93866: 2019, 31150: 6486, 38216: 9713, 24882: 6036, 59827: 7314, 69558: 4350, 30779: 4371, 73538: 325, 15555: 5211, 15814: 9762, 63688: 1450, 87882: 1432, 28624: 6252, 67795: 1189, 29012: 3334, 49365: 2490, 13399: 5454, 89075: 9685, 46432: 6054, 56684: 4598, 40304: 6531, 24561: 7191, 36082: 4864, 68243: 4224, 80501: 7536, 16248: 3695, 46931: 3702, 95869: 8534, 92926: 8369, 57301: 9762}

Solutions:

maxpop= sorted(zip\_population.values())[len(sorted(zip\_population.values()))-1]

for i in xrange(0,len(zip\_population)):

    if zip\_population.items()[i][1] == maxpop:

        print zip\_population.items()[i][0]

##alternative

pop = max(zip\_population.values())

for(zipcode, population) in zip\_population.iteritems():

    if population ==pop:

        print zipcode, population

# the value of zip code in which has maximum value

max\_zip=max(zip\_population.values())

zip\_list=[]

for (zip, maxv) in zip\_population.iteritems():

    if maxv==max\_zip:

            zip\_list.append(zip)

print zip\_list

pp = zip\_population.values()

print list

ll=zip\_population.items()

print items

for (ll, pp) in zip\_population.iteritems():

    if  pp ==max(zip\_population.values()):

        print pp, ll

zip = zip\_population.keys()

population = zip\_population.values()

max\_value = max(zip\_population.values())

for (zip,population) in zip\_population.iteritems():

        if population == max\_value:

                print "Maximum population at", zip, "is", population

temp=0

tempzip=[]

for (zip, pop) in zip\_population.iteritems():

    if pop>temp:

        temp=pop

        tempzip = [zip]

    elif pop == temp:

        tempzip.append(zip)

print "Zipcode:", tempzip, "with population of", temp

morning session with git