Welcome to The Carpentries Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

DAY 1
Uwacu Alban
Lena Karvovskaya (helper)
Kees den Heijer (instructor)
Tanya Tsui (student)
Twan Keijzer
Sebastian Fajardo
Ali Hashemi
Guillermo Mier
Giorgio Colombi
Victoria Davalos
Mariana Itzel
Srinidhi Mula
Diego Gomez
Daniela
Jaén Ocádiz
Robbert van Putten
Aihui Fu
Xiaomimg Tian
Neda Hesam
Kaiyi Zhu
Ulas Taskin
Sirasak
Shokoufeh
Mahda Foroughi
Deniz Ezgi Gulmez
Chizoba Josphine Ogugua
Hugo Kleikamp
Florence Lip
Simon Felz

Links to documents to download for the workshop: Please download them before the lesson !
for the Unix shell lesson:

- https://swcarpentry.github.io/shell-novice/data/data-shell.zip

for the Python lesson :
https://swcarpentry.github.io/python-novice-inflammation/data/python-novice-inflammation-data.zip
https://swcarpentry.github.io/python-novice-inflammation/code/python-novice-inflammation-code.zip

For now, go to www.menti.com for some interaction and introduction :-)

Windows users, start Git Bash
Mac users, start Terminal

- $ ls : list files in the current directory
- $ ks: command not found ! Shell tells you when it can't find a program you are calling
- $ pwd: print current working directory (where you are in the file system - on your hard drive)
- $ ls -F: -F indicate whether each entry is a file or directory, * indicates if it is an executable file, @ sign means it is a link
- $ ls -F /: gives us a listing of files and directories in the root directory. / means root of your directory tree
- $ ls-- help or man ls: if you are not sure about the additional options with ls. there are difference between different operating systems, that is why there are two options
    - "--help" will work on Git Bash with Windows, "man help" will work for MacOS and Linux terminal (in most cases)
    - q or control + c to return to your prompt
- $ ls -F Desktop
- $ ls -F Desktop/data-shell: to see the contents of data-shell directory
- $ cd Desktop: to change our directory to Desktop
- $ pwd: to print working directory, to confirm we are in Desktop
- $ cd data-shell: to go into data-shell directory
- $ cd data: to go into data-shell/data directory
- $ ls -F: to see the contents of data-shell/data directory
- $ cd data-shell: doesnt work, because we are in data and this is relative command and doesnt work to go up
- $ cd ..: to go one directory up
- $ pwd: to confirm we are in data-shell
- $ ls -F -a: -a stands for ‘show all’ (including hidden files); it forces ls to show us file and directory names that begin with ., such as ... . Single dot is the current location, double dot is one level up

We will learn to navigate into files and folders through **command-line interface**

- Where am I command? (present working directory)

$ pwd

- List files and folders in the current folder

$ls

- Additional information about the folder content

$ls -F

-
    - Informaiton about a command

$ls -l

- List the content of a ceratin folder (If you are in the folder that contains Desktop folder)

$ls Desktop

- List the content of a folder inside a folder
- $ls Deskto/Some_Folder
    - Change the current folder
- $cd
    - Depending on where data-shell folder is...
- $cd data-shell

- Go up one level (one folder)

$cd ..

- The length of the path is not limited to one folder

$cd Desktop/Some_folder/My_Favourite_Folder

- Create a folder called thesis

$mk thesis

- Check that the folder is there
- $ls
    - Start the text editor called nano
- $nano
    - Specify the name of the file you are creating
- $nano draft.txt
    - When exiting the editor, you can also specify the name of the file
    - Type some text into the file
- When in nano Ctrl + O saves the content into the file
- To exti type Ctrl + X
- In **command-line interface,** UP key on your keyboard shows the last command that was used. This allows to skip typing repeating commands
    - Let's rename the file
- $mv thesis/draft.txt thesis/quotes.txt
    - We changed the name of the file by "moving" it within the current folder
    - Does mv also work with folders? --> Answer: Yes, without change.
- $ls thesis
    - If everything went well you'll see the quotes.txt
    - Make sure that you are in data-shell folder
- $pwd

    - Make a copy of the file with a new name
- $cp thesis/quotes.txt thesis/quotations.txt
- How can you copy folders? Does not work without change. --> Answer: should use option -a
    - Cleaning up, removing files
- $rm thesis/quotes.txt
- $ls
    - Be careful with rm command; there is no way back
    - rm does not work for folders with content by default
    - You can remove recursively the whole folder with its content by using rm -r
- $rm -r thesis
- $ls molecules
- $cd molecules
    - Extract file names that start p
- $ls p*
- Does this also work with cp, rm, mv?? Good question, we can try :-) Answer: Works with rm, but not with mv or cp
- $ls ?ethan*
    - * is wild card, can be used for any character
    - Word count
- $ wc *.pdb: Show the number of files, words and characters => (1st) column number for lines (2nd) column number of words (3rd) column number of characters
- $wc cubane.bdb
    - Show the number of lines
- $ wc -l *.pdb
    - Write output in a file
- $ wc -l *.pdb > lengths.txt
    - > is important, it allows to work with the output of a command
    - Let's check the content of the file length
- $nano length.txt
    - Sort the file and insert the output into the original file:
- $sort length.txt
    - Numerical sort
- $sort -n length.txt
    - Let's check what sort does
- $sort --help
- $ cat lengths.txt
    - Sort the file and insert the output in the file "sorted-lengths.txt"
- $sort length.txt > sorted_length.txt
- $cp length.txt length_exp.txt
    - Reading and writing at the same time won't work. You can't use the same file as input and output
- $sort length_exp.txt > length_exp.txt
    - print Hello on the screen
- $echo Hello
    - print Hello into a file
- $echo Hello > hello.txt
- $echo Hello again > hello.txt
    - Note that the filess get owerwritten every time you use echo
- We can feed the output not to a file but to another program

- To display the small amount of flines in the file, use head
- $head cubane.pdd
    - You can limit the number of lines displayed
- $head -n 1 cubane.pbd
- $cat length.txt
    - First we do the sort command, the result will be given as input to head cmmand
- $sort length.txt | head -n 1
    - For Mac OS use man command to get information
- $man wc
    - How to combine text in the files
- $ cat hello.txt length.txt
- $ cat hello.txt length.txt > combine.txt
- cd ../creatures
- pwd
- $cat basilisk.dat
    - Let's extract classification information from multiple files
- $head -n 2 basilisk.dat | tail -n 1
    - We write a script using nano editor
- $nano classification
- for file in basilisk.dat minotaur.dat unicron.dat
- do
        - head -n 2 $file | tail -n 1
- done
- For every file in the folder
- $for file in *
- do
        - head -n 2 $file | tail -n 1
- done
- #let's go to the data_shell folder
- $cd ..
- $ls
- $bash creatures/classification
- $pwd
- In creatures/ let's modify a bit the for loop in 'classification', so that instead of going through everything that the shell find in the folder, it goes only through the files with .dat extension:
- $for file in *.dat
- do
        - head -n 2 $file | tail -n 1
- done
- Do no use spaces when naming files, as this can cause problems when running programs. Try making the for loop go through a file named "red dragon.txt". Create a file called "red dragon.txt" in your directory (where all the *.dat files are) by using:
- $nano "red dragon.txt"
- and then run the previous loop that goes through all *.dat files, and you will get a problem. All because of the space in name of the file ("red dragon.txt"). Recommendation then: use '_' instead: red_dragon.txt
- nano ../classification
- In classification let's create a backup of all the files, renaming them original-(name of file)
- $for file in *.dat

- do
- echo $file
- cp $file original-$file
- done
- Let's run the script and create all the backup files:
- $bash ../classification
- $ls
- Remember the variable called 'file' is mute. You can call it whatever:
- $for file_name in *.dat
- do
- echo $file_name
- cp $file_name original-$file_name
- done
- Recommended to add comments to your scripts to understand what you intended to do. Memory is fragile ;-)
- To add comments in a bash script use '#'. Whatever is next to the '#' will be considered as a comment and be omitted by the program.
- To refer to files (give files as input to a bash script):
- bash ../classification location_to_input_file
- In the script:
- $echo $1
- $for file_name in *.dat
- do
- echo $file_name
- done
- The $1 refers to the first input file.
- Let's do something more significant than just printing the name and location of the input file (which is saved in $1).
- nano ../classification
- In the script we write:
- $echo Going to the folder
- $echo $1
- $cd $1
- $echo Moved!
- $echo The working directory is
- $pwd
- f$or file_name in *.dat
- do
- echo #file_name
- done

$bash ../classification location_to_creatures_directory

Moving to the data-shell folder...to run the script (be aware of giving the right locations to where the 'classification' script is and provide the right path to the location_to_creatures_directory

$cd ..
$bash classification location_to_creatures_directory

What if we want give more than 1 parameter?

$ls
$nano classication
In the file let's modify the script:

- $echo Going to the folder
- $echo $1 - $2 - $3
- $echo ---parameter displayed
- $cd $1
- $echo Moved!
- $echo The working directory is
- $pwd
- f$or file_name in *.dat
- do
- echo #file_name
- done

$bash classification
Be aware we have not given any input files. So it won't print any directory name because $1 and $2 and $3 are empty.
Let's modify the script again...
$nano classification

In the script:

- $echo Going to the folder
- $echo $@
- $echo ---parameter displayed
- $cd $1
- $echo Moved!
- $echo The working directory is
- $pwd
- f$or file_name in *.dat
- do
- echo #file_name
- done

Let's run it now:
$ bash classification parameter_1 parameter_2 parameter_3

$ cd north-pacific-gyre/
$ ls
$ cd 2012-07-03/
$ ls

$ nano execution_example

- in the script:
- for datafile in $@
- do
    - echo $datafile
    - echo running goostat $datafile stats -$datafile
- done

$ bash execution_example *
$ ls

$ mkdir scripts
$ mv execution_example scripts/example
$ bash scripts/example
$ nano scripts/example
$ bash scripts/example
$ clear
(clean the screen)

$ bash scripts/example *.txt
$ ls
$ bash goostats
$ nano goostats
$ pwd
$ ls
$ nano scripts/example
(in the script)

- for datafile in $@
- do
    - echo $datafile
    - bash goostat $datafile stats -$datafile
- done

- $ bash scripts/example NENE*[AB].txt
- $ ls
- $ cd scripts
- $ ls
- $ nano scripts/example
- $ bash scripts/example NENE*[AB].txt
- $ nano scripts/example
- (in the script)
- #will run statistics on data files
- #path the program we want to call.
- GOOSTATS_PATH="/c/Users/ndintzner/.../goostats"
- for datafile in $@
- do
    - echo $datafile

    - bash $GOOSTATS_PATH $datafile stats-$datafile
- done
- $ nano scripts/example
- $ pwd
- $ cd../..
- $ cd ..
- $ cd ..
- $ pwd
- $ cd writing/
- $ ls
- $ cat haiku.txt
- $ ls
- $ grep something haiku.txt
- $ grep Thesis haiku.txt
- $ grep yesterday haiku.txt
- $ grep Yesterday haiku.txt
- (grep is case sensitive)
- $ grep --help
- $ grep yesterday haiku.txt
- $ grep -i yesterday haiku.txt
- (ingore the case sensitivity in grep)
- $ grep -i yester* haiku.txt
- Question: Can you use grep to get lines before and after a line with a match-> Answer: Yes with the option -A 5 you get 5 lines after, -B 4 get 4 lines before and -C 2 get 2 lines before and 2 lines after.
- $ ls data
- $ ls
- $ cd data
- $ ls
- $ cat one.txt
- $ cat two.txt
- $ grep Beth *
- (*: anything matches in the current directory)
- (handy to google / stack overflow: unix grep how to...)
- $ find .
- $ cd ..
- $ find .
- $ find . -type f
- (f: files)
- $ find . -type d
- (d: directories)
- $ find . -name *txt
- $ which python
- $ where python
- $ python --version
- $ grep "not the true" haiku.txt
- ==========
- Programming with Python Lesson
- ==========

- Follow the steps to install Python http://swcarpentry.github.io/python-novice-inflammation/setup/index.html
- # is used for comments
- $ is command-line command
    - the rest is code :)
- How to run python
- #There are 3 different ways:
- # -running in your terminal: python
- # -running in your terminal: ipython (interactive python)
- # iPython is the backend of Jupyter Notebooks
- # -running anaconda and launching Jupyter Notebooks (which you can also find by launching Jupyterlab for example)

Navigate into the directory where the lesson materials are located
Make sure that python-novice-inflammation-data.zip and python-novice-inflammation-code.zip are in your current directory

- $ cd data
- # Let's start jupyter notebook
- $ jupyter notebook
- # Right corner menue -> New -> Python3
    - 3+5*4 Ctrl+Enter
- # Esc - the cell turns blue
- # Create a cell below with a B
- # Shift+Enter
- # Assign a variable
    - weight_kg = 60
- # Different types of variables. We tried integer, now we do float
    - weight_kg = 60.0
- # String - be consistet about the "" or ''
    - weight_kg_text = 'weight in killograms'
    - print(weight_kg)
    - **print**(weight_kg_text, weight_kg)
- # print is a function. A function call is always done with ()
- # we can do additional calculations inside a print staitment
    - **print**('weight in pounds:', 2.2 * weight_kg)
- # Import packages/libraries
    - import **numpy**
- # Now, that nupy is imported we can use it in our commands
- # We will use it to read a csv file. Important to specify the delimiter
    - numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
- # Assign the result of reading a csv file to a variable
    - data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
    - **print**(data)
- # Find out the types of data
    - **print**(type(data)) #The output tells us that data currently refers to an N-dimensional array
    - **print**(data.dtype) # tells us that the type of the NumPy array's elements
    - **print**(data.shape) #shows the dimensions of our 'data' object, which is essentially a 2dimensional matrix of 60 rows and 40 columns
- #Working with our variable data

        - print('first value in data: ', data[0, 0])
        - **print**('middle value within data:', data[30, 20])
- #Slicing data:
    - **print**(data[0:4, 0:10])
- #The slice 0:4 means, “Start at index 0 and go up to, but not including, index 4.”
    - doubledata = data * 2.0
    - print(data[:4, :10])
    - print('original: ')
    - print(data[:4, :10])
    - print('doubledata: ')
    - print(doubledata[:4, :10])
    - tripledata = doubledata + data
    - print(tripledata)
    - **print**(tripledata[:3, 36:])
    - print(numpy.mean(data))
    - import time
    - Question can you "unimport" a package? -> Answer: del package
    - **print**(time.ctime())
- #Be careful, packages names can be owerwritten. For example time = 'now' assigns time a string 'now' if you do it, you can no longer call the library time
    - maxval, minval, stdval = numpy.max(data), numpy.min(data), numpy.std(data)
    - **print**('maximum inflammation:', maxval)
    - **print**('minimum inflammation:', minval)
    - **print**('standard deviation:', stdval)
- # If you type the name of something followed by a dot, then you can use tab completion to see a list of all functions and attributes that you can use (e.g. type numpy. and then press tab).
- # After selecting one, you can also add a question mark (e.g. numpy.cumprod?), and IPython will return an explanation of the method!
    - numpy.max?
    - patient_0 = data[0, :] *# 0 on the first axis (rows), everything on the second (columns)*
    - **print**('maximum inflammation for patient 0:', numpy.max(patient_0))
    - **print**(numpy.mean(data, axis=0).shape)
    - **print**(numpy.mean(data, axis=1))
- # We start visualizing using matpolt
    - **import matplotlib.pyplot**
    - image = matplotlib.pyplot.imshow(data)
    - matplotlib.pyplot.show()
    - ave_inflammation = numpy.mean(data, axis=0)
    - ave_plot = matplotlib.pyplot.plot(ave_inflammation)
    - matplotlib.pyplot.show()
    - max_plot = matplotlib.pyplot.plot(numpy.max(data, axis=0))
    - matplotlib.pyplot.show()
- #Grouping plots
- **import numpy**
- **import matplotlib.pyplot**
- data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
- fig = matplotlib.pyplot.figure(figsize=(10.0, 3.0))
- axes1 = fig.add_subplot(1, 3, 1)
- axes2 = fig.add_subplot(1, 3, 2)

- axes3 = fig.add_subplot(1, 3, 3)
- axes1.set_ylabel('average')
- axes1.plot(numpy.mean(data, axis=0))
- axes2.set_ylabel('max')
- axes2.plot(numpy.max(data, axis=0))

- axes3.set_ylabel('min')
- axes3.plot(numpy.min(data, axis=0))
- fig.tight_layout()
- matplotlib.pyplot.show()

Repeating Actions with Loops

- word = 'lead'
- **print**(word[0])
- **print**(word[1])
- **print**(word[2])
- print(word[3])
- word = 'tls
- **print**(word[0])
- **print**(word[1])
- **print**(word[2])
- print(word[3])

now we have an arror

- word = 'oxygen'
- **for** char **in** word:
    - **print**(char)

typical python syntax : at the end of the line and indentation
#char is a varible, its name is irrelevant. You can give it any arbitrary name

- length = 0
- **for** vowel **in** 'aeiou':
    - length = length + 1
- **print**('There are', length, 'vowels')

Assignment!!!

- x = 5
- coefs = [2, 4, 3]

y = coefs[0] * x**0 + coefs[1] * x**1 + coefs[2] * x**2
y = 97
for idx, val in enumerate(word):

    - print(idx, val)
- y = 0
- **for** idx, coef **in** enumerate(coefs):
- y = y + coef * x**idx

- Q: What if tab completion does not work ? In a Jupyter cell, type "$ conda update readline", then shift+enter to execute it. After a few seconds, the tab completion should work.

Solution to the problem : loop

for idx, coef in enumerate(coefs):

- y = y + coef * x ** idx

print(y) #to see the result

#do able without the enumerate if you use an intermediate variable, like a counter for each iteration

Info: Indentation matters! For loops, conditions, functions... everything in Python.

- Target: analysis of multiple files
- new package : glob

import glob #to get it in your environment
print(glob.glob(inflammation*.csv)) #should print all files mathcing the pattern inflammation*.csv - note: the files are NOT ordered in the output. Do NOT Expect ordered output with Glob

Recommendation: have one cell in the notebook, with all imports - it's cleaner and easier to maintain.

imports required:
import glob
import numpy
import matplotlib.pyplot

filenames = sorted(glob.glob('inflammation*.csv')) #get all files matching the pattern, and sort the list of file names
filenames = filenames[0:3] #select a subset of those
for filename in filenames:

- print(filename)
- data = numpy.loadtxt(fname=filename,delimiter=",") #loading data for the file named "filename"

- fig = matplotlib.pyplot.figure(figsize=(10.0, 3.0))
- ## COPY PASTED FROM ABOVE!
- axes1 = fig.add_subplot(1, 3, 1)
- axes2 = fig.add_subplot(1, 3, 2)
- axes3 = fig.add_subplot(1, 3, 3)
- axes1.set_ylabel('average')
- axes1.plot(numpy.mean(data, axis=0))
- axes2.set_ylabel('max')
- axes2.plot(numpy.max(data, axis=0))

- axes3.set_ylabel('min')
- axes3.plot(numpy.min(data, axis=0))
- fig.tight_layout()
- matplotlib.pyplot.show()

#output: 1 "block" per file, each "block" contains 3 graphs, and the file from which the data comes from is shown above the graph

```
odds =[1,3,5,7]
prints ('odds are', odds[3])

print('first element', odds[0])   #output: 1
print(last element', odds[3])   #output: 7
print('-1 element?',odds[-1])  #output: 7
```

-1 to start counting from the end.

```
for numbers in odds:
    print (number)

#Edits elements of a list:
names = ['Curie','Darthwing','Turing'] #typo ?
print ( "the list is : ", names)
names[1] = "Darwin"
print('final version of the list:', names)

#Edit elements of a string ?
name = "Darwin"
name[0] = d
```

==> Error! Immutable string - string does not support item assignment (assign a value to an item of the collection)

```
salsa = ['peppers','onions',"cliantro","tomatoes"] #what is cliantro ???
my_salsa = salsa
salsa[0] = "hot peppers"

print('ingredients in MY salsa', my_salsa)
```

==> expected output: both my_salsa and salsa lists contain the SAME elements
== the "=" operation on lists do NOT copy the lists, but make a reference to the list.
How to do a "copy" of the list:

my_salsa = salsa[:]

```
#comparisons and if statements:

num = 37
if num > 100 :
```

- print ("greater")

```
else:
    • print ("not grater")
print("done")


num = -3
if num > 0:
    • print (num, " is positive")
elif num = 0:
    • print (num , " is null")
else:
    • print (num , " is negative")
if (1 > 0) and (-1 > 0 ) :
    • print("both are true")
else:
    • print("at least one part is false")


max_inflammation_0 = numpy.max(data,axis=0)[0]
max_inflammation_20 = numpy.max(data,axis=0)[20]

if max_inflammation_0 == 0 and max_inflammation_20 == 20:
    • print("suspicious data!")
else:
   print("Looks OK!")

if 4>5 :
   print ("A")
elif 4 ==5:
   print ("B")
elif 4<5
   print("C")

#expected output: "C"... of course!

## Creating function

def fahr_to_celsius(temp):
   return ( (temp-32)  * (5/9) )

#to call the function (and do something)

fahr_to_celsius(32)
```

```
print ("the freezing point of water", fahr_to_celsius(32), "C")
print ("the boiling point of water", fahr_to_celsius(212), "C")

def celsius_to_kelvin(temp):
    return temp + 273.15

print("the freezing point of water in Kelvin is ", celsius_to_kelvin(0),"K")

def fahr_to_kelvin(temp):
    temp_c = fahr_to_celsius(temp)
    temp_k = celsius_to_kelvin(temp_c)
    return temp_k

print ("the boiling point of water in Kelvin is ", fahr_to_kelvin(212),"K")

#THE Function that I would use to see a given file.
def visualize (filename):
```

- print(filename)
- data = numpy.loadtxt(fname=filename,delimiter=",") #loading data for the file named "filename"

- fig = matplotlib.pyplot.figure(figsize=(10.0, 3.0))
- ## COPY PASTED FROM ABOVE!
- axes1 = fig.add_subplot(1, 3, 1)
- axes2 = fig.add_subplot(1, 3, 2)
- axes3 = fig.add_subplot(1, 3, 3)
- axes1.set_ylabel('average')
- axes1.plot(numpy.mean(data, axis=0))
- axes2.set_ylabel('max')
- axes2.plot(numpy.max(data, axis=0))

- axes3.set_ylabel('min')
- axes3.plot(numpy.min(data, axis=0))
- fig.tight_layout()
- matplotlib.pyplot.show()

```
def detect_problems(filename):
    data = numpy.loadtxt(fname=filename, delimiter=',')
    if numpy.max(data, axis=0)[0]==0 and numpy.max(data, axis=0)[20]==20:
        print('That is suspicious')
    elif numpy.sum(np.min(data, axis=0))==0:
        print('Minima add up to zero')
    else:
        print('all good!')
```

feel free to contribute :) I am not running the code, so I may tons of typos in there... I hope not.

```
##

filenames = sorted( glob.glob ( "inflammation*.csv") )
 for filename in filenames:
     print (filename)
     visualize(filename)
     detect_problems(filename)

# More function

def offset_mean( data, target_mean_value):
    return ( data - numpy.mean(data)) + target_mean_value


z = numpy.zeros((2,2))
print ( offset_mean(z,3) )

data = numpy.loadtxt( fname = "inflammation-01.csv", delimiter = ",")
print ( offset_mean (data,0) )

#does this work ? Hard to know... that's a lot of numbers

print ( "original min,mean, max are: " , numpy.min(data), numpy.mean(data), numpy.max(data) )
offset_data = offset_mean(data,0)

print ("off set data min, mean, max are : ",

    • numpy.min(offset_data),
    • numpy.mean(offset_data),
    • numpy.max(offset_data) )


#check the output, and attempt to understand what you see ;)

print ("std dev before and after: " numpy.std(data), numpy.std(offset_data)

#This offset function is non trivial. It deserve some documentation:
def offset_mean(data, target_mean_value):
    '''

      •  return a new array containing the original data with
          • its mean offset to match the desired value

    '''
    return (data - numpy.mean(data) ) + target_mean_value

# Using this format, you enable the HELP function on your new function.
#call using
help ( offset_data )
```

#You can set default value to your functions
def offset_mean(data, target_mean_value = 0.0):
    ... your function body here ...

#then you can call your function with
offset_mean(my_data) #without the 2nd parameter. 0 will be used "by default"!)

DAY 2

Florence Lip
Victoria Davalos
Yasemin Turkyilmaz (helper)
Kaiyi Zhu

Aihui Fu
Chizoba Josphine Ogugua
Deniz Ezgi Gulmez
Robbert van Putten
Giorgio Colombi
Shokoufeh Abrishami
Xiaoming Tian
Ulas Taskin
Twan Keijzer
Hugo Kleikamp
Tanya Tsui
Daniela Maiullari
Srinidhi Mula
Mahda Foroughi
Diego Gomez
Ali Hashemi
Uwacu Alban
Sirasak Tepjit
Neda Hesam
Mariana Itzel

Version Control with Git: http://swcarpentry.github.io/git-novice/

Automated Version Control: http://swcarpentry.github.io/git-novice/01-basics/index.html

Setting Up Git: http://swcarpentry.github.io/git-novice/02-setup/index.html

Go to bashk
$ cd username/Desktop (it is your own username)
$ git config --global user.name "your name" (type your own name)
$ ls -la
$ nano .gitconfig
exit nano
$ cd Desktop/
$ git config --global user.email "your email address" (type your email address, the one you will use for

github)
$ git config **--global** core. input (on macOS and Linux)
$ git config **--global** core.autocrlf true (on Windows)
$ git config -h OR
$ git config --help

Creating a Repository: http://swcarpentry.github.io/git-novice/03-create/index.html

$ cd ~/Desktop
$ mkdir swc-git
$ cd swc-git
$ ls
$ mkdir planets
$ ls
$ cd planets
$ ls
$ git init
$ ls -a (if we add the -a flag to show everything, we can see that Git has created a hidden directory within planets called .git)
$ cd .git
$ ls
$ cd ..
$ git status (to check if everything is set up correctly by asking Git to tell us the status of our project)

Tracking Changes: http://swcarpentry.github.io/git-novice/04-changes/index.html

$ cd ~/Desktop/planets
$ nano mars.txt
Type the text below into the mars.txt file:
Cold and dry, but everything is my favorite color
control X to exit
$ ls
$ cat mars.txt
$ git status
$ git add mars.txt
$ git status
Question how to un-add a file? -> Answer: git reset filename.txt
$ git diff
$ git diff mars.txt
$ git commit **-m** "Starting notes about Mars"
$ git log (lists all commits made to a repository in reverse chronological order. The listing for each commit includes the commit's full identifier, the commit's author, when it was created, and the log message Git was given when the commit was created.
$ git diff (This shows us the differences between the current state of the file and the most recently saved version)
$ nano mars.txt
Type Two moons are great, but the wolfman will not be happy
Exit nano

$ git diff
$ git status
$ git add mars.txt
$ git commit **-m** "Added mention of color and wolfman"
$ nano mars.txt
lscg
save and exit
$ git diff
$ git add mars.txt
$ git diff
$ git diff --staged (it shows us the difference between the last committed change and what's in the staging area.)
$ git commit -m "Happy mummy"
$ git log
$ git log --oneline (to reduce the quantity of information)
$ mkdir spaceships
$ ls
$ git status
$ git add spaceships
$ git status
$ cd spaceships/
$ touch apollo
$ ls
$ ls -l
$ cd ..
$ git status
$ git add spaceships/
$ git status
$ git add spaceships/apollo
$ git status
$ git commit -m "adding spaceship information - apollo"
$ ls

Exploring History: http://swcarpentry.github.io/git-novice/05-history/index.html

$ nano mars.txt
type I should not be writing this
Save and exit
$ git diff mars.txt
$ git diff HEAD mars.txtcd
$ git diff HEAD~1 mars.txt
$ git diff HEAD~2 mars.txt
(get characters with git log)
$ git diff (copy paste the first 10 characters) mars.txt
$ git status
$ git checkout HEAD mars.txt
$ cat mars.txt
$ git show (copy paste the first 10 characters)
$ git checkout (copy paste the first 10 characters) mars.txt

$ cat mars.txt

Ignoring Things: http://swcarpentry.github.io/git-novice/06-ignore/index.html

$ mkdir results
$ cd results
$ nano data.tmp
$nano data.final
$cd ..
$git add results/data.final
$ nano .gitignore
-> inside type *.tmp
$ git status
$ git add .gitignore
$ git commit -m "Added ignore list"
$ git status (should be on branch master agian with nothing to commit)

Remotes in GitHub: http://swcarpentry.github.io/git-novice/07-github/index.html

Go to github.com
Sign in
Go to your own repositories on top right and click on create a new repository
Give your repository a name in Repository name: swc-planets-(add your name)
You can choose to set it as public and private, for this exeercise we will set it as public. You might prefer to set it as private for some of your own work
Do not tick on "Initialize this repository with a README
We do not need to add now a gitignore file since we already added it, so keep it as None
Selecting a license is very important as it determines what others could do with your code. Always think ahead which license to change as it is difficult to change afterwards. For this exercise, we choose None

Then click on create repository

Now our repository is created, now we need to connect it with our local machine
Go back to git bash

$ git status

First go to Github -> copy the link, for this exercise we copy the HTTPS link

go back to bash
$ git remote add origin (copy paste the link)
(origin is a local name used to refer to the remote repository. It could be called anything, but origin is a convention that is often used by default in git and GitHub, so it’s helpful to stick with this unless there’s a reason not to.)
$ git remote -v (shows two origins, one for pushing, one for fetching)
 Now we want to push everything we have done in our local machine to our remote repository
$ git push origin master

Whenever we used git add, we transfered something from the working directory to the staging area

Whenever we did git commit, we committed all the changes from the staging area to the local repository

Now we added another level by adding a remote repository. We use git push to transfer something from local repository to remote repository

working directory - (git add) -> staging area - (git commit) -> local repository (git push) -> remote repository

$ git pull origin master (to pull changes from the remote repository to the local one)

Collaborating: http://swcarpentry.github.io/git-novice/08-collab/index.html

To be able to collaborate with someone else, you need to first invite them to the repository

Click on settings on top right
Then go to Collaborators (choose from left side menu)
Let's work in groups, one of you needs to give access to other(s)
To do so, you need to search their usernames and then click add collaborator
This way, they receive an email and they need to accept

Since we pushed, you can see all the chnages we made in our local repository in our remote repository

Now go back to your terminal
$ cd ..
$ pwd (you need to be in swc-git)
$ git clone (URL of the repository you are collaborating) (You can get this link at the greenbutton saying clone or download.)
$ ls (you should be now able to see a folder named as the repository of your neighbour)
$ cd swc-planets-(neighbour name)/
$ git status
$ git log
$ ls

Conflicts: http://swcarpentry.github.io/git-novice/09-conflict/index.html
$ nano pluto.txt
Write whatever you want
Save and exit
$ git status
$ git add pluto.txt
$ git status
$ git commit -m "(type a descriptive message here such as Added planet pluto)"
$ git push origin master
If you received an error, you should pull to get the most recent changes
$ git pull origin master
Conflicts happen all the time, it is OK, we just need to resolve this conflict
If you received an error, try to open pluto.txt
$ nano pluto.txt

You see some identifiers, this is how git shows how your version looks and how the other version looks. Our change is preceded by <<<<<<< HEAD. Git has then inserted ======= as a separator between the conflicting changes and marked the end of the content downloaded from GitHub with >>>>>>>. (The string of letters and digits after that marker identifies the commit we've just downloaded.)
Lets say we want both of them, because we dont want to lose any information. In that case, remove everything else than the two sentences. You might also prefer to keep only one of the sentences, in that case delete everything else except that sentence.
$ cat pluto.txt
$ git status

Only the person who had the conflict:
$ git add pluto.txt
$ git status
$ git commit -m "resolved conflict for pluto"
$ git push origin master
$ git log (to see all the commits)
You can also see these changes by going to your Github repository and clicking on the commit tab
Do the same exercise by inverting the roles this time

Afternoon session

Open Science presentation by Paula

Python - continued

Errors

code:

```
def favorite_ice_cream():
    ice_creams = [
        "chocolate",
        "vanilla",
        "strawberry"
    ]
    print(ice_creams[3])   # correct code: print(ice_creams[2])
favorite_ice_cream()
```

Error message:

```
    IndexError                              Traceback (most recent call last)
<ipython-input-1-70bd89baa4df> in <module>()
      6     print(ice_creams[3])
      7
----> 8 favorite_ice_cream()
<ipython-input-1-70bd89baa4df> in favorite_ice_cream()
      4         "vanilla",                                          "strawberry"
      5     ]
----> 6     print(ice_creams[3])
      7
```

- 8 favorite_ice_cream()
- IndexError: list index out of range

Syntax error:

```
def some_function() # missing a :
```

- msg = "hello, world!"
- **print**(msg)
- **return** msg

Tabs and spaces:

```
def some_function():
    msg = "hello, world!"
    print(msg)
    return msg
# python doesn't allow mix of tabs and spaces. Do not use Tabs, use space!
```

Variable name error:

```
print(a)  # a is not defined

for number in range(10):
count = count + number     # the 'count' on the right side of = has not been defined before being used.
Also, python is case senstivie.
print("The count is:", count)
```

Index errors:

```
letters = ['a', 'b', 'c']
print("Letter #1 is", letters[0])
print("Letter #2 is", letters[1])
print("Letter #3 is", letters[2])
print("Letter #4 is", letters[3])
# in order to define object as a string, ' ' is needed
```

File errors:

```
file_handle = open('myfile.txt', 'r')
```

error messages:

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-14-f6e1ac4aee96> in <module>()
----> 1 file_handle = open('myfile.txt', 'r')

FileNotFoundError: [Errno 2] No such file or directory: 'myfile.txt'
```

- file_handle = open('myfile.txt', 'w')
- file_handle.read()

error messages:

```
   ---------------------------------------------------------------------------
UnsupportedOperation                  Traceback (most recent call last)
<ipython-input-15-b846479bc61f> in <module>()
      1 file_handle = open('myfile.txt', 'w')
----> 2 file_handle.read()

UnsupportedOperation: not readable
```

Reading error messages

Codes:

```
   def print_message(day):
   messages = {
      "monday": "Hello, world!",
      "tuesday": "Today is Tuesday!",
      "wednesday": "It is the middle of the week.",
      "thursday": "Today is Donnerstag in German!",
      "friday": "Last day of the week!",
      "saturday": "Hooray for the weekend!",
      "sunday": "Aw, the weekend is almost over."
   }
   print(messages[day])
def print_friday_message():
   print_message("Friday")     # key error: Friday, can't find 'Friday' defined in 'messages'
print_friday_message()

def another_function  # missing a :
  print("Syntax errors are annoying.")
   print("But at least Python tells us about them!") # Indentation error: mismatch with other lines
  print("So they are usually not too hard to fix.")


for number in range(10):
   # use a if the number is a multiple of 3, otherwise use b
   if (Number % 3) == 0:  # 'Number' is misspelled
      message = message + a  # 'message' not defined, 'a' is not defined
   else:
      message = message + "b"
print(message)
```

Corrected codes:

```
message = ""
for number in range(10):
   # use a if the number is a multiple of 3, otherwise use b
   if (number % 3) == 0:
      message = message + "a"
   else:
      message = message + "b"
```

```
print(message)

seasons = ['Spring', 'Summer', 'Fall', 'Winter']
print('My favorite season is ', seasons[4])  #python string starts at 0

#use the try..except to handle the error
try:
    a= 5.0 / 0.0
except ZeroDivisionError:
    print('Ignoring this ridiculous results')

try:
    a= 5.0 / 0.0
except ZeroDivisionError:
    pass
b= 5.0 / 0.0

dir()  # show all the defined properties and methods

type(seasons)  # return class type of the object


c ="""
a
b
c
d
e"""
print(c) # print multiple lines
```

Defensive programming: see the error as quickly as possible

```
assert 5==5

assert 4==5  # Assertion error

numbers = [1.5, 2.3, 0.7, -0.001, 4.4]
total = 0.0
for num in numbers:
    assert num > 0.0, 'Data should only contain positive values'
    total += num
print('total is:', total)
```

Error message:

```
    ---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-19-33d87ea29ae4> in <module>()
      2 total = 0.0
```

```
      3 for num in numbers:
----> 4     assert num > 0.0, 'Data should only contain positive values'
      5     total += num
      6 print('total is:', total)

AssertionError: Data should only contain positive values
```

```
def normalize_rectangle(rect):
    '''Normalizes a rectangle so that it is at the origin and 1.0 units long on its longest axis.
    Input should be of the format (x0, y0, x1, y1).
    (x0, y0) and (x1, y1) define the lower left and upper right corners
    of the rectangle, respectively.'''
    assert len(rect) == 4, 'Rectangles must contain 4 coordinates'
    x0, y0, x1, y1 = rect
    assert x0 < x1, 'Invalid X coordinates'
    assert y0 < y1, 'Invalid Y coordinates'

    dx = x1 - x0
    dy = y1 - y0
    if dx > dy:
        scaled = float(dx) / dy
        upper_x, upper_y = 1.0, scaled
    else:
        scaled = float(dx) / dy
        upper_x, upper_y = scaled, 1.0

    assert 0 < upper_x <= 1.0, 'Calculated upper X coordinate invalid'
    assert 0 < upper_y <= 1.0, 'Calculated upper Y coordinate invalid'

    return (0, 0, upper_x, upper_y)
```

```
a
```

```
Error messages:
    ---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-3-325036405532> in <module>
----> 1 print(normalize_rectangle( (4.0, 2.0, 1.0, 5.0) )) # X axis inverted

<ipython-input-1-c94cf5b065b9> in normalize_rectangle(rect)
      6     assert len(rect) == 4, 'Rectangles must contain 4 coordinates'
      7     x0, y0, x1, y1 = rect
----> 8     assert x0 < x1, 'Invalid X coordinates'
      9     assert y0 < y1, 'Invalid Y coordinates'
     10

AssertionError: Invalid X coordinates
```

Test Driven Development:

```
assert range_overlap([ (0.0, 1.0) ]) == (0.0, 1.0)
assert range_overlap([ (2.0, 3.0), (2.0, 4.0) ]) == (2.0, 3.0)
assert range_overlap([ (0.0, 1.0), (0.0, 2.0), (-1.0, 1.0) ]) == (0.0, 1.0)
```

Error messages:

```
   ---------------------------------------------------------------------------
NameError                 Traceback (most recent call last)
<ipython-input-25-d8be150fbef6> in <module>()
----> 1 assert range_overlap([ (0.0, 1.0) ]) == (0.0, 1.0)
      2 assert range_overlap([ (2.0, 3.0), (2.0, 4.0) ]) == (2.0, 3.0)
      3 assert range_overlap([ (0.0, 1.0), (0.0, 2.0), (-1.0, 1.0) ]) == (0.0, 1.0)

NameError: name 'range_overlap' is not defined
```

(Back to python command line in Git bash. Jupyter notebook is handy for small program)

```
$ cd ..
$ ls
$ cd code
$ ls
$ python readings_01.py
```

If you have issue with using numpy, try to re-open Git Bash (right click: log in as admin). Remember to check the right path/folder to try it again.

```
$ cat readings_0
$ cat reading_01.py
$ ls
$ nano readings_01.py
```

In script:

```
import sys
import numpy

def main():
    script = sys.argv[0]
    filename = sys.argv[1]
    data = numpy.loadtxt(filename, delimiter=',')
    for m in data.mean( axis=1):
        print(m)

 main()
```

$ python readings_01.py

$ nano readings_01.py

```
import sys
import numpy

def main():
    script = sys.argv[0]
    filename = sys.argv[1]
    data = numpy.loadtxt(filename, delimiter=',')
    for m in data.mean( axis=1):
        print(m)

print("sys.argv is", sys.argv)
```

$ python readings_01.py
$ python readings_01.py toto some more somemore

$ nano readings_01.py

```
import sys
import numpy

def main():
    script = sys.argv[0]
    try:
        •  filename = sys.argv[1]dp
        •  data = numpy.loadtxt(filename, delimiter=',')
        •  except:
            • print("input arguments are not correct")
            • return
    for m in data.mean( axis=1):
        print(m)

print("sys.argv is", sys.argv)
main()
```

$ python readings_01.py
$ python readings_01.py toto some more somemore

$ python readings_01.py ../data/inflammation-01.csv

$ ls ../data/
$ ls ../
$ ls ../
$ ls ../data/

$ nano readings_01.py

```
import sys
import numpy

def main():
    script = sys.argv[0]
    try:
        filename = sys.argv[1]
        data = numpy.loadtxt(filename, delimiter=',')
    except:
        print("input arguments are not correct")
        return
    process(data)
#   for m in data.mean( axis=1):
#       print(m)

def process(data):
    for m in data.mean(axis=1):
        print(m)

print("sys.argv is", sys.argv)
main()
```

$ python readings_01.py ../data/inflammation-01.csv
$ ls
$ ls ..
$ ls ../data
$ ls ../data/*small*
$ pwd
$ python readings_01.py ../data/small-01.csv
$ cat ../data/small-01.csv
$ python readings_01.py ../data/inflammation-01.csv
$ nano readings_01.py

```
import sys
import numpy

def main():
    script = sys.argv[0]
    action = sys.argv[1]
    try:
        filename = sys.argv[2]
        data = numpy.loadtxt(filename, delimiter=',')
    except:
        print("input arguments are not correct")
        return
```

- assert action in ['--min', '--max', '--mean'], "Action is not valid (should min or max or mean)"
- process(data)

```
#   for m in data.mean( axis=1):
#       print(m)

 def process(data):
     for m in data.mean(axis=1):
         print(m)

def do_math(a,b):
    return a+b

print("sys.argv is", sys.argv)
main()
```

$ nano test_readings_01.py

```
def test_do_math():
```

- """
- This fuction is to test the Do Math function in "readings_01.py"
- """
- test_return = do_math(1,2)
- assert test_return == 3, "do_math failed"

```
test_do_math()
```

$ python test_readings_01.py
$ cat readings_01.py
$ nano test_readings_01.py

```
from readings_01 import do_math

def test_do_math():
```

- """
- This fuction is to test the Do Math function in "readings_01.py"
- """
- test_return = do_math(1,2)
- assert test_return == 3, "do_math failed"

```
test_do_math()
```

$ python test_readings_01.py
$ cat readings_01.py
$ nano readings_01.py

```
import sys
```

```
import numpy

def main():
   script = sys.argv[0]
   action = sys.argv[1]
   try:
      filename = sys.argv[2]
      data = numpy.loadtxt(filename, delimiter=',')
   except:
         print("input arguments are not correct")
         return
   assert action in ['--min', '--max', '--mean'], "Action is not valid (should min or max or mean)"
   process(data)
#   for m in data.mean( axis=1):
#      print(m)

def process(data):
    for m in data.mean(axis=1):
       print(m)

def do_math(a,b):
   return a+b

print(__name__)

if  __name__ == '__main__':
   main()
```

```
$ python readings_01.py --min as
$ python test_readings_01.py
```

Rileva
linguaAfrikaansAlbaneseAmaricoAraboArmenoAzeroBascoBengaleseBielorussoBirmanoBosniacoBulgaroCatalanoCebuanoCecoChichewaCinese sempCinese tradCoreanoCorsoCreolo HaitianoCroatoCurdoDaneseEbraicoEsperantoEstoneFilippinoFinlandeseFranceseFrisoneGaelico scozzeseGalizianoGalleseGeorgianoGiapponeseGiavaneseGrecoGujaratiHausaHawaianoHindiHmongIgboIndonesianoIngleseIrlandeseIslandeseItalianoKannadaKazakoKhmerKirghisoLaoLatinoLettoneLituano LussemburgheseMacedoneMalayalamMaleseMalgascioMalteseMaoriMarathiMongoloNepaleseNorvegeseOlandesePashtoPersianoPolaccoPortoghesePunjabiRumenoRussoSamoanoSerboSesothoShonaSindhiSingaleseSlovaccoSlovenoSomaloSpagnoloSundaneseSvedeseSwahiliTagikoTailandeseTamilTedescoTeluguTurcoUcrainoUngheresеUrduUsbecoVietnamitaXhosaYiddishYorubaZulu

AfrikaansAlbaneseAmaricoAraboArmenoAzeroBascoBengaleseBielorussoBirmanoBosniacoBulgaroCatalanoCebuanoCecoChichewaCinese sempCinese tradCoreanoCorsoCreolo HaitianoCroatoCurdoDaneseEbraicoEsperantoEstoneFilippinoFinlandeseFranceseFrisoneGaelico scozzeseGalizianoGalleseGeorgianoGiapponeseGiavaneseGrecoGujaratiHausaHawaianoHindiHmongIgboIndonesianoIngleseIrlandeseIslandeseItalianoKannadaKazakoKhmerKirghisoLaoLatinoLettoneLituano LussemburgheseMacedoneMalayalamMaleseMalgascioMalteseMaoriMarathiMongoloNepaleseNorvegeseOlandesePashtoPersianoPolaccoPortoghesePunjabiRumenoRussoSamoanoSerboSesothoShonaSindhiSingaleseSlovaccoSlovenoSomaloSpagnoloSundaneseSvedeseSwahiliTagikoTailandeseTamilTedescoTeluguTurcoUcrainoUngheresеUrduUsbecoVietnamitaXhosaYiddishYorubaZulu

La funzione vocale è limitata a 200 caratteri

Opzioni : Cronologia : Opinioni : DonateChiudere

G
M
T
Y

Rileva
linguaAfrikaansAlbaneseAmaricoAraboArmenoAzeroBascoBengaleseBielorussoBirmanoBosniacoBulgaroCatalanoCebuanoCecoChichewaCinese sempCinese tradCoreanoCorsoCreolo HaitianoCroatoCurdoDaneseEbraicoEsperantoEstoneFilippinoFinlandeseFranceseFrisoneGaelico scozzeseGalizianoGalleseGeorgianoGiapponeseGiavaneseGrecoGujaratiHausaHawaianoHindiHmongIgboIndonesianoIngleseIrlandeseIslandeseItalianoKannadaKazakoKhmerKirghisoLaoLatinoLettoneLituano LussemburgheseMacedoneMalayalamMaleseMalgascioMalteseMaoriMarathiMongoloNepaleseNorvegeseOlandesePashtoPersianoPolaccoPortoghesePunjabiRumenoRussoSamoanoSerboSesothoShonaSindhiSingaleseSlovaccoSlovenoSomaloSpagnoloSundaneseSvedeseSwahiliTagikoTailandeseTamilTedescoTeluguTurcoUcrainoUngheresеUrduUsbecoVietnamitaXhosaYiddishYorubaZulu

AfrikaansAlbaneseAmaricoAraboArmenoAzeroBascoBengaleseBielorussoBirmanoBosniacoBulgaroCatalanoCebuanoCecoChichewaCinese sempCinese tradCoreanoCorsoCreolo HaitianoCroatoCurdoDaneseEbraicoEsperantoEstoneFilippinoFinlandeseFranceseFrisoneGaelico scozzeseGalizianoGalleseGeorgianoGiapponeseGiavaneseGrecoGujaratiHausaHawaianoHindiHmongIgboIndonesianoIngleseIrlandeseIslandeseItalianoKannadaKazakoKhmerKirghisoLaoLatinoLettoneLituano LussemburgheseMacedoneMalayalamMaleseMalgascioMalteseMaoriMarathiMongoloNepaleseNorvegeseOlandesePashtoPersianoPolaccoPortoghesePunjabiRumenoRussoSamoanoSerboSesothoShonaSindhiSingaleseSlovaccoSlovenoSomaloSpagnoloSundaneseSvedeseSwahiliTagikoTailandeseTamilTedescoTeluguTurcoUcrainoUngheresеUrduUsbecoVietnamitaXhosaYiddishYorubaZulu

La funzione vocale è limitata a 200 caratteri

Opzioni : Cronologia : Opinioni : DonateChiudere

G
M
T
Y

Rileva
linguaAfrikaansAlbaneseAmaricoAraboArmenoAzeroBascoBengaleseBielorussoBirmanoBosniacoBulgaroCatalanoCebuanoCecoChichewaCinese sempCinese tradCoreanoCorsoCreolo HaitianoCroatoCurdoDaneseEbraicoEsperantoEstoneFilippinoFinlandeseFranceseFrisoneGaelico scozzeseGalizianoGalleseGeorgianoGiapponeseGiavaneseGrecoGujaratiHausaHawaianoHindiHmongIgboIndonesianoIngleseIrlandeseIslandeseItalianoKannadaKazakoKhmerKirghisoLaoLatinoLettoneLituano LussemburgheseMacedoneMalayalamMaleseMalgascioMalteseMaoriMarathiMongoloNepaleseNorvegeseOlandesePashtoPersianoPolaccoPortoghesePunjabiRumenoRussoSamoanoSerboSesothoShonaSindhiSingaleseSlovaccoSlovenoSomaloSpagnoloSundaneseSvedeseSwahiliTagikoTailandeseTamilTedescoTeluguTurcoUcrainoUngheresUrduUsbecoVietnamitaXhosaYiddishYorubaZulu

AfrikaansAlbaneseAmaricoAraboArmenoAzeroBascoBengaleseBielorussoBirmanoBosniacoBulgaroCatalanoCebuanoCecoChichewaCinese sempCinese tradCoreanoCorsoCreolo HaitianoCroatoCurdoDaneseEbraicoEsperantoEstoneFilippinoFinlandeseFranceseFrisoneGaelico scozzeseGalizianoGalleseGeorgianoGiapponeseGiavaneseGrecoGujaratiHausaHawaianoHindiHmongIgboIndonesianoIngleseIrlandeseIslandeseItalianoKannadaKazakoKhmerKirghisoLaoLatinoLettoneLituano LussemburgheseMacedoneMalayalamMaleseMalgascioMalteseMaoriMarathiMongoloNepaleseNorvegeseOlandesePashtoPersianoPolaccoPortoghesePunjabiRumenoRussoSamoanoSerboSesothoShonaSindhiSingaleseSlovaccoSlovenoSomaloSpagnoloSundaneseSvedeseSwahiliTagikoTailandeseTamilTedescoTeluguTurcoUcrainoUngheresUrduUsbecoVietnamitaXhosaYiddishYorubaZulu

La funzione vocale è limitata a 200 caratteri

Opzioni : Cronologia : Opinioni : DonateChiudere