**Manipulating files**

We know how to read to, write to, and append to files, but we may want to copy, move, rename, or delete files. We may want to process many files within a directory. Two key modules for this process are os (short for “operating system”) and shutil (short for “Shell UTILities”).

To create a directory

import os

os.mkdir ("Sequences")

#to make multiple directories, use os.mkdirs()

#os.mkdirs ("Folder1/Folder2/Sequences")

Please download the sequences on courselink to this directory, and make a new file, “temp.” In unix touch temp works. Or

open ("./Sequences/temp","w") *#this makes a file*

We can see what is in a directory:

import os  
#os.listdir("."). #for current directory  
for name in os.listdir("./Sequences"):

print("Directory element name is " + name)

Files can be selected based on conditions.

import os

open ("temp")

for name in os.listdir("."): #this can be any directory

if name.endswith("txt"): #the condition

print (name)

We can remove a file using the os.remove command

os.remove("./Sequences/temp") #removes the file

os.rmdir("Empty\_directory") #also works

We can rename or copy a file using the rename function.

os.rename("./Sequences/temp","./Sequences/temp2") #renames

os.rename("./Sequences/temp2","temp") #moves and renames

os.rename works on folders too.

To copy a file, one can use:

shutil.copy("source","target")

To copy a directory, one can use shutil.copytree("My/Path/A","My/Path/B")

We can test if a path or folder exists with os.path.exists() that returns a TRUE or FALSE

print(os.path.exists("./Sequences/temp/seq\_1.fasta"))

print(os.path.exists("./Sequences/seq\_1.fasta"))

**Running a program and recovering its output**

To run a program, the program should already be in your path. The functions for running programs reside in the subprocess module. A “subprocess” is a program that is started by another program.

import subprocess

subprocesses.call("Path/To/Executable")

Any output is printed straight to the screen.

subprocess.call("/bin/date")

subprocess.call("/bin/ls")

subprocess.call("/bin/date +%B",shell=True) #takes arguments

More often, we want to capture the program output.

subprocess.check\_output() # has the same arguments as subprocess.call

current\_month=subprocess.check\_output("/bin/date +%B",shell=True)

b'November\n' is the output. The b stands for “bytes.” One can process this information to make is readable.

current\_month.decode('ascii') #makes a string

print(current\_month.decode('ascii'))

**User input**

Most programs will use command line arguments to get information about how they should run. We can ask for user input with raw\_input(). One often uses rstrip() on raw\_input().

answer=raw\_input("enter a number please")

print ("The number is " + answer)

As we already discussed, command line arguments passed to the program can be recovered with importing the sys module. The elements of “sys.argv” are the command line elements provided by the user.