Programming for Bioinformatics

Saul Pierotti

December 8, 2019

Course organization

- 5/12 mid term exam on paper, it is worth 8/32 points, finals are on February and march
- This module pf the course is about Python

Linux an CS basics

- The main resource of a computer are RAM and CPU
- The OS allocates resources to programs, and impedes interference among them
- Linux is a kernel, it manages resources for the OS
  - It derives from Unix, like also MacOS
  - It is multithreaded (it can run more than 1 program at the same time) and multiuser
  - The users are isolated from each other, they cannot interfere
- A Linux distro includes an install system for the distro itself, drivers, a package manager, tools
  - Some tools are really specific, and probabily I will never need them
  - A package manager allows to install and remove tools
- One user is root, the superuser
  - It should be used only when needed and with extreme caution
- The shell is the main interface of the OS
- The directory structure is a rooted tree
  - The root directory in linux is called `/`
- A file is the name given to a set of data
- A file needs to be contained in a directory
- An extension is an indication of the filetype, but does not determine it
- Python programs are text files with extension .py
- Some basic shell commands
  - The `Tab`button autocompletes the commands in the shell
  - `cd` is used to enter in a directory
  - `mkdir` creates a directory
  - `ls` lists all files and directories inside the current directory
    - `ls -l` gives more informations on the file, e.g. permissions
  - `.` is a shortcut for the current dir
  - `..` is a shortcut for the parent dir
  - `~` is a shrotcut for the home dir of the current user
  - Starting my path with the root dir `/` makes the path absolute
  - Starting with a directory directly is like starting with `./`
  - Writing `cd` without parameters is like writing `cd ~`
  - `pwd` prints the current dir
  - `cp` and `mv` are the copy and move commands
  - `rm` permanently deletes files
- Files can have different permissions

  - Users can be selectively allowed to read, write and/or execute files
  - The owner of a file can set permissions for everyone, users and groups
  - Permissions can be read as `-rwxrwr--`, where the first 3 charachters refer to user, then group, then others, then all
  - Permissions are added and remove with the `chmod` command
  - `chmod u+x` adds execute permission to user, while `chmod o-w` removes write permisission to others
  - `chmod 777` gives full permissions to everyone
- A running program is identified by a unique PID (program id)
  - `ps -a` lists all the running processes with PID
  - `kill PID` kills the running process
    - This should be used only as a last option, I risk to lose data (!)
- A single program can span multiple processes
- `sleep` suspends the current shell for the specified time
- `man` opens the manual of a command
- The argument of a command is the subject of the operation
- The parameters of a command are the option that specify its action
- The output of a command can be written to a file with the redirect operator `>`
  - `ps -a > output.txt`
  - The error messages will still be printed on screen
  - If the file already exists, it is overwritten (!)
  - If I use the append operator `>>` instead, it adds the output at the end of the file
- To redirect errors we use `2>`, while to redirect both normal output and error `&>` is used
- To show the content of a file we can open it in a text editor, or use the following commands
  - `head` shows the first 10 lines
  - `tail` shows the last 10 lines
  - `cat` shows the whole file, and it is unpractical with long files
  - `more` shows the whole file page by page
  - `less` shows the whole file allowing to scroll
- There is a plethora of text editors
  - `nano` and `pico` are easy to use
  - `vim` and `emacs` are more advanced but less easy
  - Some have a GUI version, like `gvim` and `xemacs`
- A computer is fast, but stupid, It does exactly what you tell it to do
- Programming is useful for dealing with complex operations, repetitive tasks, huge amounts of data
- Sometimes I can do things with a PC without knowing how they are done: libraries
- Documentation is really useful as a reference, but is not really good for learning
- Things work most of the time, until they do not work once and no one knows why
- I can run a script in the background using an & after the command
  - `~ python test.py &`

Python basics

- Pyhton is fast to implement, widely used, has many libraries, it is well documented
  - It is not used so much by computer scientists, but a lot by non computer scientists
  - We want to be as efficient as needed, not as efficient as possible (!)
    - Using C can improve efficiency, but not as much as writing a good program
  - It is an imperative language
  - We can use C when we need to work really low-level, but often it is not needed
- A Python program can be written and used in different ways
  - I can create a text file and run it from the shell
  - I can use an IDE to write, execute, access documentation
  - I can write directly in the Python interpreter without creating a file
- The operator / is division in python2, while in python 3 the operator // is integer division

  - The result of integer division is itself an integer
  - Floating point division in python3 is done by the operator /
  - In python2 the type of division is determined by the context
    * It is integer division only if both numbers are integers, otherwise it is floating point
    * If I want to do floating point division between integers, I need to first convert one of them to a float
- The operator % is the remainder of an integer division
- The operator * is multiplication, ** is exponentiation, + and - are sum and addition
- There are many built-in functions to perform calculations
- `e` and `pi` are recognised as the respective constants
- To use functions from libraries I need to use the syntax
  - `from math import *`
    * `*` is a wildcard that means everything
    * I can also import a single command
  - The need to import commands is due to avoid an enormous number of function name clashes when I define a custom function
- A variable is the name of a memory location that can store a value
- Strings can be accessed by charachter by putting the index in parentheses
  - `str[0]`
- Substrings can be extracted as
  - `str[2:5]` extracts 2 included until 5 excluded
  - If I omit beginning or end, it considers the beginning or the end
  - `str[-2:2:-1]` specific to go by jumps of -1 (go backwards)
- String concatenation is done with the operator +
- You cannot change a string by assigning a value to an element of the string, you need to create a new string
- User input is collected with `input("message")` in python3 and `raw_input("message")` in python2
- Some methods for strings
  - `s.upper()`
  - `s.lower()`
  - `s.replace("a","b")`
  - `s.startswith("a")`

Functions

- A function is a code block with a name
- A built-in function is readily understood by the python interpreter

```
def fun(par):
    my code+par
x = "some data"
fun(x)
```

- The return statement assigns its value to the function
- In the following x=1

```
def fun():
    return 1
x = fun()
```

- The first statement in a function is called docstring

```
def fun():
    """this is a function that doesn't do anything"""
```

- Comments in pyhton are made with # and they are useful to make code more readable

Lists

- Lists behave similarly to strings
- To check if an element is in a string or list I can do

```
my_list = [1,2,3,4]
print(2 in my_list)
>>> True
```

- The main difference is that in lists I can reassign elements
- The split function splits a string in a list separated by the separator given as an argument
- I cannot split for the empty charachter, to separate a string in any charachter I should use `list(str)`

For loop

```
my_list = [1,2,3,4]
for num in my_list:
    print(num)
>>>1
>>>2
>>>3
>>>4
```

If statements

```
if test:
    code
elif test2:
    code
else:
    code
```

- Logical tests are `==, !=, <, >, <=, >=`
- Logical operations are `and, or, not`

Files

- A file can be opened in reading (r), write (w) and append (a)
  - Opening in write destroys the prevoious content of the file (!)

  ```
  filein = open("path/to/file","r")
  print filein
  >>><open file 'path/to/file', mode 'r' at 0x00000>
  ```
- If I use a for loop in a file, I loop through its lines
- By default, \n is included in the line and it can be removed with string.rstrip()
- A file can be closed with filein.close, and this is really important when we are writing in a file
  - Writing operations can be put in a buffer by the OS, and so if my program crashes I do not now if the file has actually been written
- To write in a file I can do `fileout.write("some string")`
  - The \n has to be added manually (!)

Dictionaries

- A dictionary is made of values, that can be retrieved through keys

```
my_dic = {key1:value1, key2:value2}
print(my_dic[key1])
>>>value1
```

- Key and value can be strings, integers, floats
- The number of key-value pairs in the dictionary can be retrieved with `len(my_dic)`
- Dictionaries are mutable

```
D1 = {"name":"saul"}
D1["surname"] = "pierotti"
print(D1)
>>>{"name":"saul,"surname":"pierotti"}
```

- In general, variables can be eliminated with del

```
x = 1
del x
print(x)
>>>Traceback (most recent call last):
>>>  File "<stdin>", line 1, in <module>
>>>NameError: name 'x' is not defined
```

- Also dictionary entries can be eliminated

```
D1 = {"name":"saul"}
D1["surname"] = "pierotti"
del D1["name"]
print(D1)
>>>{"surname":"pierotti"}
```

- The get method of dictionaries allows to return the value of a key specified as first argument, or the second argument if the key does not exist

```
D1 = {"A":9}
print(D1.get("B",1))
print(D1.get("A",2))
>>>9
>>>1
```

- This is useful for incrementing a value, or creating it if it does not exist

```
my_seq = "ATTTAATGGGCCCGGCCCGGG"
for char in my_seq:
    D1[char] = D1.get(char, 0) + 1
print(D1)
>>>{"A":3,"T":4,"C":6,"G":8}
```

- The keys method returns a type dict_keys object with the keys of a dictionary
  - In python2 it returns a normal list

  ```
  D1 = {"name":"saul"}
  print(D1.keys())
  >>>dict_keys(["name"])
  ```
- By default dictionary elements are not sorted, they are in order of insertion
  - In python3 they do not have an order
- I can sort a list with the sort method

```
my_list = [1,3,56,21,12]
my_list.sort()
print(my_list)
>>>[1,3,12,21,56]
```

- The system time in python can be retrieved with

```
import datetime
datetime.datetime.now()
```

- I can get the execution time of a task by subtracting the current time at the beginning and end of it