Taller de Herramientas Computacionales

Elías Jiménez Cruz

27/Enero/2019

Contents

Introducción

Este compendio es el conjunto de las bitácoras elaboradas por cada día de la materia. Se pretende que con esto se logre un resumen de todo lo visto en el curso, obteniendo así una guía para futuros trabajos y proyectos.

Chapter 1

Bitácora 1

A continuación realizaremos una breve descripción de lo visto durante la primera clase. Vimos primero lo que es un sistema operativo, un sistema que administra el hardware de una computadora y que permite la interacción entre éste y el usuario. Existen varios sistemas operativos, entre los que destacan Windows, MacOs, Android y iOs cada uno creado para distintos propósitos y distintas arquitecturas de computadora. Sin embargo, uno de los más singulares hasta el momento es un sistema operativo llamado Linux, creado por Linus Torvalds, el cual es un sistema de código libre, es decir, un sistema en el que cualquier persona, sin un estricto ánimo de lucro, puede tener libre acceso al código del sistema para así mejorarlo y colaborar en su desarrollo. En consecuencia, al no ser necesario un ánimo de lucro, este tipo de sistemas operativos pueden ser usados por cualquier usuario sin tener que pagar necesariamente una licencia por él. De esta forma, Linux puede ser un sistema operativo gratuito, que es respaldado por miles de desarrolladores que día a día hacen de esta plataforma una mejor opción frente a otros sistemas.
Ahora bien, como se dejó entrever, una de las características de Linux, que puede ser una de sus principales ventajas o uno de sus mayores inconvenientes, es que, al tener muchísimos desarrolladores, los objetivos o intereses perseguidos a la hora de desarrollar la plataforma no siempre es homogénea, lo que se traduce en el surgimiento de una gran variedad de versiones de este sistema, cuya única propiedad en común es el kernel, o el código interno sobre el cual se monta la interfaz gráfica, las características y las aplicaciones. Cada una de estas versiones es conocida como una distribución de Linux, que va a tener ciertas características que la harán especial, como puede ser su gran poder de procesamiento, su bajo consumo de recursos, su gran versatilidad para manejar distintos tipos de hardware, etc. No obstante, si bien hay una gran variedad de distribuciones, es sabido

que existen tres distribuciones principales, de las cuales se derivan las demás: Ubuntu, Debian y Fedora. Estas versiones son sistemas operativos fuertes por sí mismos, aunque al ser tan versátiles no satisfacen todas las necesidades de todos los usuarios, pues para aquellos que deseen un sistema ligero que les permita adentrarse poco a poco al entorno de Linux, estos sistemas no son la mejor opción. Un ejemplo de este último tipo de sistema es Slax, un sistema operativo basado en Debian que apenas si consume 100 MB de memoria en su estado más básico, lo cual hasta lo vuelve ideal para usarse con las PC más antiguas. Entonces, hay una versión de Linux para todos los intereses de los usuarios.
Algo a tener en cuenta es que las distribuciones de Linux son sistemas operativos que, en mayor o menor medida, requieren de cierto conocimiento en el manejo de su Shell interno, que es un intérprete de comandos, un programa que está a la espera de una instrucción para ejecutarla. Gráficamente es una pantalla negra en la que se teclean los comandos y se navega únicamente con el teclado, un proceder que evoca al manejo de las computadoras en los principios de los años ochenta. En Linux, este Shell se conoce como Bash y sus comandos sirven para administrar y utilizar el sistema, así como para navegar por sus directorios y visualizar sus archivos. Si bien la lista de comandos disponibles puede ser demasiado extensa, podemos resumirla en los comandos más promientes o esenciales:

1. set: Muestra todas las variables de entorno del sistema, las cuales están en mayúsuculas. Una variable de entorno es aquella que determina una parte específica del sistema.
2. pwd: Indica la ruta del directorio en que alguien se encuentra.
3. cd "directorio": Traslada la ubicación al directorio escogido.
4. touch "archivo": Verifica la existencia de un archivo, y si no existe lo crea.
5. ls: Muestra los archivos y carpetas del directorio actual.
6. ./ "archivo": Ejecuta un archivo desde el directorio actual.
7. "Nombre de intérprete" "programa": Para correr un programa se usa esta estructura, primero se escribe el nombre del intéprete bajo cuyo lenguaje el programa a correr fue escrito, para entonces dejar un espacio y escribir el nombre del programa deseado.
8. df -lh: Sirve para mostrar las particiones del sistema.
9. top sirve para mostrar información del procesador y los núcleos.
10. apt-get update: Sirve para actualizar el comando apt-get, con el cual se pueden descargar aplicaciones desde su biblioteca.

11. apt-get install "nombre de aplicación": Instala en el sistema la aplicación deseada.

12. chmod xxx "archivo": Cambia los permisos de lectura, escritura y ejecución de un archivo a nivel usuario, grupo y general. En cada x se modificarán los permisos respectivamente para los niveles mencionados, y se modificarán en base al siguiente criterio: Para activar la lectura del archivo se sumará cuatro a la casilla del nivel deseado, si se desea activar la escritura, se sumará dos y si se quiere activar la ejecución se sumará uno, dando un máximo de siete y un mínimo de cero para cada nivel.

Así, con estos comandos se puede checar información general del sistema, navegar a través de los directorios, crear y administrar archivos, así como descargar aplicaciones.

Chapter 2

Bitácora 2

En la clase 2 finalmente dejamos las charlas y en el taller de ciencias de la computación 1 comenzamos a trabajar en el entorno de Linux. Los que trajimos nuestro propio computador, instalamos el sistema operativo y ejecutamos los comandos "apt-get update" y "apt-get upgrade" para actualizar la biblioteca de donde se descargan aplicaciones. A continuación debimos realizar un paso importante para la dinámica del curso, que consistió en crear un repositorio público en el servidor git con el objeto de subir nuestros trabajos para su revisión. El procedimiento es sencillo en realidad, primero hay que crear la cuenta en github.com, con nuestro número de cuenta como nombre de usuario y nuestro correo de ciencias como el correo de la cuenta. Una vez creada la cuenta, se tiene que acceder a nuestros repositorios en github.com y entonces hacer clic en nuevo. Entonces se pedirá la confirmación de nuestro correo, con lo que se nos enviará un mensaje al correo para así verificarlo. Una vez hecho esto, se nos permitirá crear el repositorio, sólo habrá que nombrarlo. En mi caso particular, lo nombré como TallerHC. Ahora viene la parte más complicada, que es sincronizar ese repositorio con el Linux de nuestro computador. En Bash, hay que escribir el comando "apt-get install git", que instalará la aplicación de git. A continuación, hay que escribir "git init", que iniciará la aplicación, para entonces darnos de alta en la aplicación con nuestro correo y nombre de usuario, para lo cual escribimos los comandos "git config –global user.email "correo"" y "git config –global user.name "NombreDeUsuario"". Una vez registrada la cuenta que queremos sincronizar, tenemos que clonar el repositorio de git que ya creamos en el servidor (TallerHC), en la carpeta de nuestra preferencia. Para ello, vamos a github.com y nos dirigimos al repositorio recién creado, donde encontraremos una opción que dice "clonar o descargar". Damos clic ahí y copiamos la url que se nos presenta. Ahora nos dirigimos en Bash a la carpeta deseada y

escribimos el comando "git clone "DirecciónQueCopiamos"" con lo que el repositorio completo se descargará a la carpeta deseada. Con esto estaremos preparados para empezar a guardar información en la carpeta descargada y así sincronizarla con el repositorio en git. Una vez se hayan guardado archivos en la carpeta TallerHC, para subirlos al servidor hay que ir en Bash a la carpeta de nuestro repositorio y escrbir "git add *" para que los archivos nuevos que no se hayan sincronizado previamente se preparen para la subida. Entonces hay que hacer un comentario de lo que vamos a subir, para ello escribimos el comando "git commit", donde se nos presentará una sección para escribir la reseña. En el caso de Slax, se presiona F2 para guardar el comentario y F10 para salir de la sección. Una vez hecho esto, se procederá a subir los nuevos archivos, con el comando "git push". El sistema indicará si se realizó bien el proceso o no. Ahora bien, se puede checar previamente si existen archivos pendientes de sincronizarse con el servidor, para lo cual se utilizará el comando "git status" el cual indica si hay archivos que no han sido sincronizados y si estos ya están en el proceso de sincronizarse. En el caso de que no haya, simplemente indicará que el árbol del directorio se encuentra limpio. Por otra parte, es posible que lo que se quiera hacer es descargar los archivos del servidor para actualizar el directorio que tenemos en nuestra computadora. Para ello, se puede usar "git status" para comprobar si existen archivos por descargar y en el caso afirmativo usamos "git pull" para la sincronización de descarga.
De esta manera, en esta clase nos familiarizamos más en el entorno de Linux, y si bien hubo problemas para seguir los pasos en la ejecución de las actividades, el simple hecho de seguirlos fue de mucha utilidad para dejar de tener como extraño a un sistema que es usado por muchísimos desarrolladores por su característica intrínseca de seguridad, versatilidad y apertura de la información.

Chapter 3

Bitácora 3

Se checaron una vez más los pasos para crear y sincronizar un repositorio de git. Como ejercicio se crearon en la carpeta TallerHC la carpeta Clases, que contendrá los archivos creados durante el curso y que estará dividida en dos carpetas más: Latex y Programas. La primera carpeta contendrá nuestras bitácoras de cada día y los archivos de Latex que creemos, mientras que programas contendrá los documentos para Python que creemos por igual. Para esta creación se hizo uso del comando "mkdir" en Bash, donde se hizo la observación que si se escribía "mkdir -p "ruta de directorios" se podía crear más de una carpeta por comando, estableciendo una ruta parental. A continuación se creó un archivo de nombre Clase03.txt que se guardó en la carpeta de Latex, en el cual se anotaron los pasos para crear un repositorio en git. Para hacer este archivo se utilizó el comando vi en Bash, el cual lanza la aplicación Vim, un editor de texto plano que se encuentra dentro del mismo Shell. El comando a ejecutar desde la carpeta de Latex es "vi Clase03.txt" el cual abre Vim para la edición del texto. Mostrará el archivo vacío y para comenzar a editarlo es necesario teclear la letra "i". Se escribe el documento y una vez finalizada la edición se pulsa la tecla "esc" para dejar de editarlo. Ahora, para guardar y salir, se debe escribir el doble punto ":" seguido de "wq" (sin comillas) para entonces pulsar la tecla enter. El sistema regresará a Bash, y se podrá comprobar que el documento creado existe en la carpeta actual. Se puede usar el comando "cat Clase03.txt" para checar de forma sucinta lo escrito. Ahora bien, si se desea salir sin guardar, se debe escribir ":q!", con lo cual el sistema no guardará los cambios y, en su caso, no creará el archivo. A continuación, se debió subir los directorios y el nuevo archivo al repositorio de git, para ello se usaron los comandos "git add *", "git commit" y "git push", tal y como se describió en la bitácora anterior.
Después de esta práctica, se procedió a explicar un poco el proced-

imiento para resolver un problema, un paso esencial para programar en cualquier lenguaje de computadora. Básicamente, primero hay que definir el problema a resolver, es decir, hay que entenderlo debidamente, o de otra manera no sabríamos siquiera qué es lo que hay que resolver. A continuación es preciso analizarlo y delimitarlo, hay que entederlo en base a su contexto y sus fundamentos. Después llega el momento de buscar soluciones al problema, buscar posibles aproximaciones, de forma vaga e intuitiva, que nos alumbrarán un poco el camino a seguir. Esto sólo será un primer paso, pues ahora que se tienen las posibles soluciones se necesita descrbirlas con detalle, en aras de hacerlas consistentes y comprobar si son efectivas en realidad. Así, una vez que se tiene una solución específica, llega la hora de dar un paso más y volverla general, que no sólo resuelva el problema en algunas circunstancias, sino que sea útil en muchas situaciones para su mejor implementación. De esta manera, se tendrá un solución fuerte y resistente en la práctica.

Chapter 4

Bitácora 4

En la cuarta clase, comenzamos a ver el entorno de Python. La versión de Python que debemos usar es Python 2, en mi caso, utilizaré la versión 2.7.13. Muchas distribuciones de Linux tienen preinstalada alguna versión de Python, para combrobarlo basta con ir a Bash y escrbibir "python" como comando y presionar dos veces la tecla tabulador. Ahí se mostrarán todas las versiones de Python ya instaladas en el sistema. Ahora bien, lo que está instalado es el lenguaje únicamente, no se encuentra instalado algún editor adecuado para hacer scripts de programas escritos en ese lenguaje. Si bien es posible escrbir un programa de Python en el editor de texto preinstalado en el sistema, es recomendable usar un editor adecuado, cuya interfaz sea amigable para escribir un programa, es decir, que tenga elementos que permitan detectar errores, completar paréntesis o comandos, resaltar con colores determinados comandos y palabras reservadas para distinguirles fácilmente. En el caso de la clase, se utilizará el editor Idle, que es un IDE o entorno de desarrollo integrado, que a fin de cuentas es una aplicación que proporciona servicios para facilitar el desarrollo de software. Para instalar Idle, se utilizó el comando "apt install idle", el cual se encargó de todo el proceso. Ahora bien, es posible correr Python desde Bash, basta con ejecutar el comando "python", pero con este método se accede al intérprete de una forma muy básica, a nivel consola, no muy apto para escribir y guardar scripts. En vez de ello, simplemente se va a ejecutar el comando idle, el cual abrirá una ventana con la consola de Python, pero desde la que se podrán abrir archivos nuevos y existentes, editarlos, guardarlos y ejecutarlos cómodamente desde una interfaz gráfica. Así, una vez conocido nuestro nuevo entorno de trabajo, pasamos a averiguar cómo resolver un problema desde la perspectiva vista de la clase pasada. El problema en cuestión fue cómo calcular la posición de una pelota en un momento determinado, cuando es lanzada a una velocidad inicial hacia arriba. Esto es claramente un problema que se

resuelve con la fórmula de la caída libre, donde la posición deseada se expresa como:

$$Posicion = (VelocidadInicial)(tiempo) - \frac{(gravedad)(tiempo^2)}{2} \tag{4.1}$$

¿Pero cómo se haría un programa en Python que calcule, por ejemplo, la posición del objeto cuando la velocidad inicial es $34\frac{m}{s}$, en el segundo cinco después de que fuera lanzado y considerando a la aceleración de la gravedad como $9.81\frac{m}{s^2}$? Para ello tuvimos que aprender el uso en Python de las operaciones básicas, que son multiplicación, suma, división, resta y potencia, en vistas de poder escribir la fórmula deseada en un programa. A continuación, en Idle creamos el programa Ejemplo1Pelota.py mediante la herramienta Nuevo archivo del menú Archivo y lo guardamos en la carpeta de Programas, con lo cual escribimos el programa declarando el valor de las variables que representarían a la velocidad inicial, a la gravedad y al tiempo, y las combinamos en la fórmula deseada, cuyo valor lo asignamos a la variable posición, que sería la que sería imprimida por el programa mediante el comando print. Así, se guardó el programa y se presionó la tecla f5 para que fuera ejecutado en idle, el cual arrojó un resultado incorrecto, producto de un pequeño detalle que hay que tener en cuenta a la hora de programar en esta versión del lenguaje: el programa arroja los resultados conforme al tipo de variable que se le da. Es decir, si en una operación se le ordena sumar dos números enteros, el programa arrojará como resultado un número entero. Esto representa un problema a la hora de dividir, pues si se le ordena dividir dos números enteros, el programa arrojará un número entero, sin importar si el resultado no lo era. Para corregir este problema, basta con ingresar alguno de los números que componen la división como un número de tipo flotante, es decir, como un número que tenga decimales, aunque estos sean cero. Python distingue a 2 como número entero y a 2.0 como número flotante, simplemente es el tipo de variable en el que se guarda la información. De esta manera, se corrigió el error en el programa y en esta ocasión arrojó el resultado deseado al ejecutarlo. Resultó un gran avance en el curso, al grado de que hasta se dejó como tarea hace un programa que resolviera el cálculo de alguna otra fórmula matemática. En mi caso, escogí el teorema de Pitágoras.

Chapter 5

Bitacora 5

Retomando lo visto y hecho la clase anterior, aprendimos primero un uso más elaborado de la instrucción print de Python. Para ello vimos un nuevo tipo de variable, que es la variable cadena, o string. Esta variable almacena texto, guarda una secuencia de caracteres sin algún valor lógico. La manera de declarar una cadena es poner el texto a declarar entre comillas, es decir, "Esto es una cadena.". Se puede guardar con algún nombre específico, tal y como se guarda una variable de tipo entero. Ahora bien, con el comando print, se puede mandar a imprimir una cadena sin mayores inconvenientes, sólo se escribe print "cadena". Esto es útil en programas complejos, para solicitar información o mostrar información que ayude a los usuarios a navegar por el sistema. Sin embargo, las cadenas también pueden volverse útiles en programas simples, en el momento en que se desea editar el resultado de algún cálculo hecho en el programa. En vez de sólo imprimir el número que da daterminada operación, con una cadena se puede mandar a imprimir el número con algún comentario, que lo haga entendible a aquél que lo lea. En el programa hecho en la clase anterior, con una cadena resulta sencillo mandar a imprimir la posición del objeto en el segundo cinco de una forma legible: "La posición de la pelota tras 5 segundos de haber sido lanzada es...", lo cual además le confiere cierta elegancia. Para lograr esto es necesario hacer uso de cierto comodines que se colocan en los lugares de la cadena donde queremos imprimir las variables. Se utiliza el símbolo % seguido de una letra que indica el tipo de variable que desea imprimirse. A continuación, al final de la cadena, se insertan las variables a imprimir, en el orden en que se fueron colocando sus comodines respectivos y en el formato %(variable1, variable2, etc...). Ahora bien, respecto a estas letras, hay muchas y diversas, pero para mencionar algunas %E es para formato científico, %s para cadena, %g para imprimir la variable en su expresión más corta posible, %f

para imprimir una variable en una expresión flotante, usualmente se usa de la forma %a.bf, donde a es el número mímimo de caracteres a imprimir y b el número de decimales a imprimir. Si a no está determinado, se imprime el valor de la variable sin alterar. También tenemos %e para formato científico de números pequeños o %d para número enteros. Se realizó un programa, de nombre EjemplosPrint.py para mostrar más comodines y sus usos. Ahora, como se dejó entrever, es posible imprimir variables de algún tipo determinado en un formato específico, como en el caso de las variables flotantes, aunque el mencionado archivo ya se encarga de mostrar más ejemplos. Así, finalmente retornamos al caso del programa de la caída libre, y haciendo un nuevo programa de nombre Ejemplo1Pelota_05.py se editó el resultado mediante una cadena. El código que imprime la cadena es: "print 'La posición de la pelota en el t=%g es %.2f' %(t,y)".
Finalmente pasamos a ver una función diferente de Python: Las funciones. Éstas son una manera de crear "acciones" en Python que pueden invocarse en un momento deseado, sin tener que volver a implementar el código completo. Por ejemplo, en el caso del programa de caída libre, puede implementarse una función que calcule la posición del objeto no sólo para valores fijos, sino para los valores que deseemos sin tener que escribir la fórmula en todo momento. Para esto se usa el comando def seguido del nombre que deseemos poner a la función, seguido de las variables que serán usadas para hacer nuestro cálculo entre paréntesis y separadas por comas, todo seguido por dos puntos. Entonces nosotros debemos escribir debajo nuestra operación, guardándola en una variable que será devuelta por la función con el comando return. la variable devuelta por "return" determinará el tipo de variable que será otorgado a la función, y podrá utilizarse como si de una variable normal se tratará. Así pues, en el programa de la caída libre, podemos escribir su función de la siguiente manera:

```
def CaidaLibre(tiempo, VelocidadInicial):
    posicion = v0*t - 1.0/2*9.81*t**2
    return(posicion)
```

Esto permitirá introducir los valores que deseen para el tiempo y la velocidad inicial, simplemente con invocar la función por su nombre e incluir en el orden establecido los valores entre paréntesis. Si se corre el archivo una vez, incluso idle es capaz de invocar la función desde donde sea que se haya guardado el archivo.
El anterior sin duda fue un gran avance, dejándose como tarea el implementar una función de la tarea pasada. Sin embargo, la clase aún no terminaba y se comenzó a ver Latex, que es un sistema de composición de textos, que permite crear documentos de alta calidad y que es usado por muchos matemáticos para escribir sus libros o

los resultados de sus investigaciones. Para ello, al igual que Python, se instaló un editor de textos con el comando apt, de nombre texstudio, y en mi caso se instaló el lenguaje de Latex bajo el nombre de texlive, pues muchas otras distribuciones de Linux ya incluyen el lenguaje por defecto. Ahora bien, creamos un documento nuevo de nombre Ejercicio01.tex, con el que aprendimos la estructura básica de todo documento de Latex, así como algunos comandos de edición. Aprendimos los comandos para comenzar el documento, para crear un título, para que el sistema reconozca caracteres de español, para poner negritas o cambiar de color el texto y para enumerar un listado, lo cual será esencial para crear las bitácoras del curso.

Chapter 6

Bitácora 6

En python se vio el comando if, el cual permite que el sistema ejecute una instrucción con determinadas condiciones. Básicamente la estructura es if (condición): y debajo se escribe la instrucción con identación. Como ejemplos calculamos el valor absoluto de un número, en el que si era negativo lo devolvía positivo y si era positivo lo devolvía igual. Como ejercicio se dejó hacer diana.py y como tarea una extensión del mismo. En Latex vimos como poner secciones y símbolos matemáticos.

Chapter 7

Bitácora 7

Comenzamos a ver el comando while, el cual permite que una instrucción se repita varias veces hasta que deje de cumplirse una condición determinada. Básicamente se escribe while(condición): y debajo se pone de forma indentada la instrucción. Este comando es demasiado útil si se sabe utilizar. Se pueden realizar sumas de una sucesión de números, restas, e incluso se pueden poner con condicionales if, aumentando su poder. De esta forma, hicimos el ejercicio Ulam.py, en vistas de practicar. Usualmente se utiliza un contador en estos comandos, el cual lleva la cuenta de las veces que se repite la instrucción, y al llegar a un número determinado de veces hace que se detenga el ciclo. No obstante, la condición no necesariamente tiene que ser de naturaleza númerica, puede ser un valor booleano, es decir, de verdadero o falso, el cual es determinado desde dentro del ciclo mismo. Esto permite que las posibilidades se extiendan, y sea diferente a otras opciones de ciclos, los cuales se verán más adelante.

Chapter 8

Bitácora 8

Volvimos a checar el ciclo while e if, con los cuales se dejaron 10 ejercicios de tarea a realizar. El primero era calcular el MCD de dos números, el segundo era calcular el tiempo en que se encuentra el objeto que es lanzado en el ejercicio de tiro vertical de la pelota, el tercero era convertir de grados Celsius a Farenheit y viceversa, el cuarto era dada una posición en la sucesión de Fibonacci, calcular su valor, el quinto era calcular la suma de los primeros n números, el sexto era calcular el promedio de 10 números, el séptimo era calcular el mismo promedio de diez números, pero también determinar el número más pequeño y el más grande, y por último, los últimos tres teníamos que idearlos nosotros, siempre y cuando tuvieran ciclo while e if.
Después vimos una nueva manera de imprimir valores en cadenas. Una copia del código la introduciré aquí:

```
'''Elías Jiménez Cruz
409085596
Taller de herramientas computacionales
{aquí va una descripción del programa y lo que hace}'''

x= 10.5
y= 1.0/3
z= 15.3
#x,y,z= 10.5,1.0/3,15.3
H= '''El punto en R3 es (x,y,z) = (%.2f,%g,%G)''' %(x,y,z)
print H

G= '''El punto en R3 es (x,y,z) = ({laX:.2f},{laY:g},{laZ:G})''' .format(laX=x,laY=y,laZ=z)
print G

#import math as m
```

```
#from math import sqrt
#from math import sqrt as s

from math import *
x = input("Dame el número del cual quieres conocer su raíz: ")
print "La raíz cuadrada de %.2f es %f" %(x,sqrt(x))
```

Chapter 9

Bitácora 9

Se revisaron los ejercicios encargados en la tarea del día anterior. Se aclararon dudas y dieron consejos. Se vio un nuevo ejercicio, en el cual se definía una función en la que se ingresaba un número, y si era par se dividía entre 2 y si era impar se multiplicaba por 3 y se le sumaba 1, tras lo cual se repetía la operación hasta que finalmente el número ingresado se redujera a 1. Esto claramente es una práctica de los comandos while e if. Aquí está la transcripción del código del ejercicio:

```
def updown(x):
i = 0
while x != 1:
if x%2 == 0:
x=x/2
else:
x=3*x+1
i=i+1
return("Las veces que el proceso se repitio para que el numero se redujera a 1 fueron: %d" %(i))
```

También checamos cómo importar comandos desde el sistema operativo para ser usados en Python. Sólo se trata de escribir en Idle el comando "from os import "comandos"", con los cuales se podrán utilizar en Linux desde Python. Después de esto vimos Latex, en el cual checamos arreglos como tablas, alineamientos y matrices. He aquí una transcripción:

```
\section*{Matrices}
%\dots puntos suspensivos
%\vdots puntos suspensivos verticales
\[
\begin{bmatrix}
x_{2} & x_{3}\\
```

```
x_{4} & x_{6}
\end{bmatrix}
\]
\[
\begin{bmatrix}
x_{2} & x_{5} & \dots\\
x_{5} & x_{20} & \dots\\
\vdots & \vdots & \vdots
\end{bmatrix}
\]
$\sum$
\section*{Tablas}
\[
\begin{array}{|c|c|c|}
\hline
f(t) & F(s) & \mbox{remark}\\
\hline\hline
\delta(t) & 1 & \mbox{impulse function}\\
u(t) & \frac{1}{5} & \mbox{unit step function}\\
e^{at}u(t) & \frac{1}{s-a} & \mbox{one-side exponential }\\
\hline
\end{array}
\]
\section*{Alineamiento}
\begin{align*}
2x - 5y &= 8\\
2x - 9y &= -12\\
\end{align*}
```

Chapter 10

Bitácora 10

En la clase 10 empezamos a ver listas en Python, que son una estructura de datos bastante importante para este lenguaje de programación. Si ha de definirse en lenguaje coloquial de alguna manera, es un conjunto ordenado, el cual consta de elementos que tienen una posición en dicho conjunto. Para construirlo, basta con poner los elementos en el orden deseado, separados por comas, y encerrarlos dentro de corchetes. "[Esto, es, una, lista, de, 7, elementos]". Puede asignársele un nombre y guardarla como si se tratase de una variable cualquiera. Al fin y al cabo es un objeto, al igual de una cadena o un número entero, sólo que con propiedades diferentes. Ahora bien, divergiendo un poco en el tema, existe un comando llamado bool, el cual permite determinar si una variable de algún tipo determinado está vacía o guarda algún valor relacionado con nulidad, dependiendo de su tipo de variable. Si no es nulo, devuelve verdadero; si lo es devuelve falso. Entonces, en el caso de números enteros, será nulo su valor si es cero, en el caso de cadenas, será nulo si la cadena está vacía. En el caso de las listas, si efectivamente está vacía la lista, es decir, si es igual a []. Ahora bien, también se checó otro comando aplicable únicamente a las listas, el comando len, el cual devuelve el número de elementos de una lista en particular. Es decir, si ponemos len([a,b,c,d]), len será igual a 4. La anterior es la forma de invocar la instrucción. También vimos algunos métodos de las listas, el .append("valor"), que agrega un elemento a la cola de la lista, el .insert(posicion, valor), en cual agrega en la posición deseada de la lista un valor dado, desplazando a la derecha el valor que ocupaba esa posición. También vimos .pop() y .pop(posicion) el cual quita, en el primer caso, de la lista el último elemento o, en el segundo caso, el elemento cuya posición se determina. Este valor quitado se puede guardar como variable. Se vio el método .extend("lista") en el cual a una lista dada se le pueden agregar los elementos de otra lista, que quedarían a la cola de la primera. Ahora, la manera en que

se determinan las posiciones de los elementos de una lista es simple, el primer elemento será el elmento 0, y el último será el elemento len(lista)-1, lo que indica que todos los elementos se enumerarán del 0 al len(lista).1. Si se quiere imprimir o guardar un elemento determinado de una lista, basta con escribir lista[posicion], lo cual regresará el elemento solicitado. Ahora bien, es posible meter una lista dentro de otra, como si fuera un elemento cualquiera, por lo que si se quiere imprimir o guardar un elemento de esa lista anidada, basta con poner lista[posicionDeLaListaAnidada][Elemento], lo cual devolverá el valor sin problemas.//Ahora, se pasó a ver otro ciclo de Python, el ciclo for, el cual permite que se repita un proceso por cada elemento de una lista determinada. He aquí una transcrpición de los códigos vistos en clase:

```
{C = -20
iC = 5
gradosC = []
while C <= 40:
F = (9.0/5)*C +32
print C, F
#C = C + iC
gradosC.append(C)
C += iC
print '\n'

print '   c    F'
for grado in gradosC:
F=(9.0/5)*grado+32
print '%5d %5.1f' %(grado, F)
print '\n'

indice = 0
print '   c    F'
while indice < len(gradosC):
C = gradosC[indice]
F=(9.0/5)*C+32
print '%5d %5.1f' %(C, F)
indice += 1
print '\n'

gradosC=[]
for C in range(-20,45,5):
gradosC.append(C)
print gradosC
print '\n'
```

```
gradosC=[]
for i in range(0,21):
C=-20 + i*2.5
gradosC.append(C)
print gradosC
print '\n'}
```

Chapter 11

Bitácora 11

Se prosiguió revisando el tema de listas y el ciclo for. Se checaron dudas de la tarea y se revisaron nuevos temas. En particular, se dejó como tarea hacer un programa que calculara la suma superior y la inferior de un conjunto de áreas dado por la función -6x3+5x2+2x+12, evaluada en n intervalos cuyas distancias entre sí serían multiplicadas por la función de cada borde, a manera de calcular sumas de Rieman finitas. Para ello, el usuario debería ingresar los intervalos deseados y el intervalo de la función a evaluar.
También se pasó a checar en Latex el uso de los documentos libro, en el que checamos cómo poner índices, cómo hacer capítulos y poner bibliografía, secciones y vínculos con hyperref. He aquí una transcripción:

```
\documentclass{book}
%\usepackage[spanish]{babel}
\usepackage[utf8]{inputenc}
%\usepackage{biblatex}
\usepackage{hyperref}

\title{Taller de Herramientas Computacionales}
\author{Elías Jiménez Cruz}
\date{17/Enero/2019}

\begin{document}
\maketitle
%Aquí inicia el índice del contenido del texto.
\tableofcontents
\section*{Introducción} Este libro es para fortalecer
el conocimiento de la materia de Taller de Herramientas
Computacionales.
\url{www.google.com}
```

```
\hyperref[Google]{www.google.com}

%Aquí inician los capítulos del libro.
\chapter{Uso básico de Linux}
\section{Distribuciones de Linux}
\section{Comandos}
\chapter{Introducción a LateX}
\chapter{Introducción a Python}
\begin{verbatim}
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-

'''Elías Jiménez Cruz
409085596
Taller de herramientas computacionales
{aquí va una descripción del programa y lo que hace}'''

x= 10.5
y= 1.0/3
z= 15.3
#x,y,z= 10.5,1.0/3,15.3
H= '''El punto en R3 es (x,y,z) = (%.2f,%g,%G)''' %(x,y,z)
print H

G= '''El punto en R3 es (x,y,z) =
({laX:.2f},{laY:g},{laZ:G})'''
.format(laX=x,laY=y,laZ=z)
print G

#import math as m
#from math import sqrt
#from math import sqrt as s

from math import *
x = input("Dame el número del cual quieres conocer su raíz: ")
print "La raíz cuadrada de %.2f es %f" %(x,sqrt(x))
end{verbatim}
\input{prueba.py}
\section*{Orientación a objetos}

\begin{thebibliography}{9}
%\bibitem{Computación}
Autor blah!\\
\textit{cualquier cosa}
blah!
\end{thebibliography}
```

```
\end{document}
```

Chapter 12

Bitácora 12

En esta clase seguimos reforzando el tema de listas y for. En especial, pasamos a checar cómo hacer una lista directamente con un ciclo for, sin tener que escribir todo los elementos. Básicamente sólo se escribe primero la condición o fórmula que va a tener cada elemento en términos de lo segundo que se escribe, que es una lista (que simplemente puede ser un range()). Así, si se quiere crear una lista de grados Farenheit desde una lista de grados Celsius basta con determinarla así: [i*1.8 + 32 for i in gradosC]. También podría ser posible hacer una lista de listas de esta manera, simplemente determinar primero el formato para después poner los elementos a ingresar en el formato: [[C,F] for C,F in zip(gradosC,gradosF)], donde zip es la función que permite discernir los elementos de las dos listas. Vimos cómo es posible guardar los elementos de una lista sin tener que declarar uno a uno los elementos a guardar, basta con que escribamos lista[x:y] para imprimir los elementos de lista desde la posición x a la y-1. Es posible hacer esto con las sublistas, sólo basta con poner el segundo índice. Para finalizar Python, realizamos el documento tablas, cuya transcripción es la siguiente:

```
n=12; gradosC=[-5 + i*0.5 for i in range(n)]
gradosF=[i*1.8+32 for i in gradosC]

ListaCombinada=[[C,F] for C,F in zip(gradosC,gradosF)]
print ListaCombinada
```

Ahora pasamos a ver en Latex el uso de Beamer, que es una especie de powerpoint, un tipo de documento que permite hacer presentaciones en forma de diapositivas. Vimos como abrir diapositivas nuevas, como pasar de una diapositiva a otra con transblindshorizontal, cómo poner títulos en la diapositiva con frametitle y como poner temas predeterminados. He aquí la transcricpión de lo visto:

```
\documentclass{beamer}
\usepackage{graphicx}
\usepackage[utf8]{inputenc}
%\usepackage[spanish]{babel}
\graphicspath{{../Imagenes/}}
%\usetheme{Antibes}
%\usetheme{AnnArbor}
%\usetheme{Berkeley}
%\usetheme{CambridgeUS}
%\usetheme{Goettingen}
%\usetheme{Hannover}
%\usetheme{Ilmenau}
%\usetheme{Berlin}
%\usetheme{Boadilla}
%\usetheme{Darmstadt}
\usetheme{Bergen}

\def\insertauthorindicator{¿Quién?}
\def\insertdateindicator{Fecha}
\title{Taller de Herramientas Computacionales}
\author{Elías Jiménez Cruz}
\date{\today}

\begin{document}
\maketitle
\begin{frame}
%\transboxin
\transblindshorizontal
\frametitle{Mi primera presentación en LaTeX}
\begin{center}
\includegraphics[scale=0.40]{EscudoFC.png}
\end{center}
\end{frame}
\begin{frame}
\frametitle{Segunda diapositiva}
Esta es mi segunda diapositiva
\end{frame}
\begin{frame}[fragile]
\begin{verbatim}
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-

'''Elías Jiménez Cruz
409085596
Taller de herramientas computacionales
{aquí va una descripción del programa y lo que hace}'''
```

```
x= 10.5
y= 1.0/3
z= 15.3
#x,y,z= 10.5,1.0/3,15.3
H= '''El punto en R3 es (x,y,z) = (%.2f,%g,%G)''' %(x,y,z)
print H

G= '''El punto en R3 es (x,y,z) =
({laX:.2f},{laY:g},{laZ:G})'''
.format(laX=x,laY=y,laZ=z)
print G

#import math as m
#from math import sqrt
#from math import sqrt as s

from math import *
x = input("Dame el número del cual quieres conocer su raíz: ")
print "La raíz cuadrada de %.2f es %f" %(x,sqrt(x))
end{verbatim}
\end{frame}
\end{document}
```

Chapter 13

Bitácora 13

En esta clase se comenzó a ver un tema importante en la programación, que es el tema de la recursión. Básicamente, una función recursiva es aquella que se utiliza a sí misma para definirse. Con esto, se puede realizar un ciclo a manera de while o de for, sólo que de una manera que no recurre a gastar memoria con un comando de ciclo. Ciertamente es un tema difícil de entender al principio, pero aquí hay algunos ejemplos:

```
{# -*- coding: utf-8 -*-
'''Programe una función que determine si dos listas son iguales.
Dos listas se consideran iguales si tienen igual longitud y sus elementos
en cada índice también lo son.'''

def determinaIgualdad(l1, l2):
if len(l1) == len(l2):
if l1 == [] and l2 == []:
return "Si son iguales."
else:
if l1.pop() == l2.pop():
return determinaIgualdad(l1, l2)
else:
return "No son iguales."
else:
return "No son iguales."
}
```

```
'''Realice un programa recursivo que determine el factorial de un número natural n ingresado.'''
def recursion(n):
if n == 0:
return 0
elif n == 1:
return 1
```

```
else:
return n * recursion(n-1)

n = input('''Dame el valor de n cuyo factorial quieres conocer: ''')
print "El factorial de %d es: " %(n),
print recursion(n)
```

Como puede apreciarse, todo programa de recursión requiere de una "base" el cual es que detiene el proceso. En el programa de factorial, por ejemplo, al ingresar n, el proceso hace que n se multiplique por la siguiente ejecución de la función, que sería n-1, y a su vez n-1 se multiplicaría por n-2 y así sucesivamente hasta llegar a 1. Al llegar a 1, el proceso devolvería la multiplicación de todas las funciones ejecutadas, dando así el factorial.

Chapter 14

Bitácora 14

En la penúltima clase, se checaron dudas sobre la tarea del día anterior, que consistió en realizar los ejercicios de la sección de evaluación del libro Python fácil. Eran ocho ejercicios, de los cuales el que presentó más dudas fue el del laberinto. Consiste en lo siguiente: "Diseñe y programe un algoritmo recursivo que encuentre la salida de un laberinto, teniendo en cuenta que el laberinto se toma como entrada y que es una matriz de valores True, False, (x,y), (a,b), donde True indica uno bstáculo; False, una celda por la que se puede caminar; (x,y), el punto donde comienza a buscarse la salida y (a,b), la salida del laberinto.". Entonces, se checaron varias posibles soluciones, yo en particular realicé el algoritmo de la siguiente manera:

```
{# -*- coding: utf-8 -*-
'''Diseñe y programe un algoritmo recursivo que encuentre la salida
de un laberinto, teniendo en cuenta que el laberinto se toma como entrada y
que es una matriz de valores True, False, (x,y), (a,b), donde True
indica uno bstáculo; False, una celda por la que se puede caminar;
(x,y), el punto donde comienza a buscarse la salida y (a,b), la salida del
laberinto.'''

def laberinto(matriz, e):
try:
if matriz[e[0]-1][e[1]] == False:
matriz[e[0]][e[1]] = True
return(laberinto(matriz, (e[0]-1,e[1])))
except:
pass
try:
if matriz[e[0]][e[1]+1] == False:
matriz[e[0]][e[1]] = True
return(laberinto(matriz, (e[0],e[1]+1)))
```

```
except:
pass
try:
if matriz[e[0]+1][e[1]] == False:
matriz[e[0]][e[1]] = True
return(laberinto(matriz, (e[0]+1,e[1])))
except:
pass
try:
if matriz[e[0]][e[1]-1] == False:
matriz[e[0]][e[1]] = True
return(laberinto(matriz, (e[0],e[1]-1)))
except:
pass
return("La salida es: "+str(e))}
```

Esta es el archivo de implementación, el cual explica en qué consiste y como debe introducir los datos el usuario5:

```
{# -*- coding: utf-8 -*-
from Ejercicio05 import laberinto

interruptor = True
while interruptor == True:
matriz= input('''
Vamos a avergiuar cuál es la salida de un laberinto formado por una matriz.
Para ello, dame primero una matriz boolenana, sin espacios entre los elementos,
cuyas filas y columnas formarán las celdas por las que pasará el camino. Si el
valor de la celda es True, se le considerará un bloque por el que no se
podrá pasar, si es False, se podrá pasar por ahí. Así, las celdas que tengan
el valor false deberán formar un camino entre las celdas True, pero sin que
alguna celda False toque lateralmente a más de otra celda False.
Así mismo, una vez ingresado el laberinto, presiona enter e ingresa en
forma de tupla las coordenadas de filas y columnas donde se ubicará la entrada
del laberinto, que será la celda donde se empezará a recorrer el camino hasta el
final. Para dar mayor realismo al ejercicio, procura que la celda de entrada y
la de salida sean adyacentes a los bordes de la matriz:
''')
e=input()
print laberinto(matriz,e)
interruptor = bool(raw_input("Si desea repetir el proceso ingrese cualquier
caracter, \nde lo contrario presione enter: "))
}
```

Chapter 15

Bitácora 15

Al final, en esta última clase, se realizó una evaluación, que consistió en una primera parte de 30 preguntas hechas en el classroom de google, de opción múltiple, en una segunda que consistió en escribir un texto con temas vistos que no estaban en el examen, y por último en un examen de 7 ejercicios para realizar en Python, los cuales debían subirse en un carpeta llamada práctica a github.