**Universidad Nacional Autónoma de México**

Facultad de Ingeniería

**Laboratorios de computación**

**salas A y B**

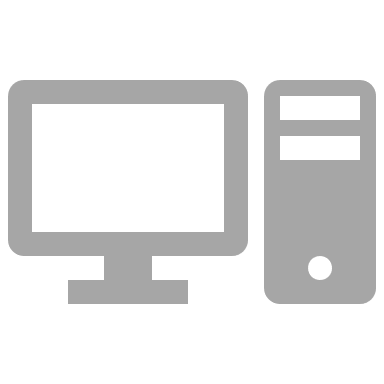
**PROFESOR:** M.I. Marco Antonio Martínez Quintana

**ASIGNATURA:** Estructura de Datos y Algoritmos I

**GRUPO:** 17

**NO DE PRÁCTICA:** 10

**NOMBRE:** Reyes Mendoza Miriam Guadalupe

**SEMESTRE:** 2020-2

**FECHA DE ENTREGA:** 14/04/2020

**OBSERVACIONES:**

**CALIFICACIÓN:**

**INTRODUCCIÓN A**

**PYTHON (II)**

**OBJETIVO**

Aplicar las bases del lenguaje de programación Python en el ambiente de Jupyter notebook.

**INTRODUCCIÓN**

**¿QUÉ ES PHYTON?**

Python es un lenguaje de programación orientado a objetos, es decir, un idioma que la computadora conoce y nos sirve para ordenarle distintas acciones.

Pero Python no es sólo un lenguaje de programación, sino además es *interpretado*, esto quiere decir que la computadora por sí sola no conoce el lenguaje y lo comprende, si no que necesita un *intérprete*. Esto sería similar a visitar un país del cual no conocemos ni una palabra de la lengua nativa, pero viajamos con una persona que conoce nuestra lengua y la de ese país, y nos hace de traductor para que logremos entendernos con las demás personas.

El mundo de la programación se divide en dos clases de lenguajes: ***interpretados*** y ***compilados***. La diferencia es que, en un lenguaje compilado, el programador escribe el código en el lenguaje de su preferencia, siempre y cuando sea un lenguaje que se compile, y luego, por medio del compilador, ese código se “traduce” al lenguaje ensamblador que entiende la computadora. La ventaja de esto es que el código ya se encuentra totalmente traducido y su ejecución es veloz.

En cambio, con un lenguaje interpretado el intérprete traduce el código para que la computadora lo comprenda a medida que se va ejecutando, y esto puede llegar a resultar un poco más lento en la ejecución que un código compilado.

Una de las principales ventajas es que se puede ir probando el código a medida que lo vamos escribiendo, un intérprete no sabe ni le importa cuando termine el código para hacer su trabajo. En cambio, un compilador realiza su tarea hasta que encuentra la instrucción de *fin*, por lo que, si el código que queremos compilar no está completo, no lograremos compilar y ejecutar nuestro programa. Esta diferencia hace que programar en un lenguaje interpretado sea mucho más dinámico, y de esta forma se optimiza el tiempo de programación y depuración del código escrito.

**ESTRUCTURAS DE CONTROL**

Un bloque condicional o estructura de control, es un bloque de código que se ejecuta sí y solamente si la instrucción de control que lo contiene ve cumplida su condición cuando uno de los bloques se ejecuta en seguimiento de las instrucciones condicionales se termina.

Nada evita que un bloque se encuentre dentro de una función de un método de un módulo, o incluso de una clase. Una estructura de control Puede ubicarse en bucle o incluso dentro de otra estructuras de control.

Una condición es la evaluación de una expresión que transforma de forma determinista dicha expresión por uno de los dos valores ***True*** o ***False***.

Las condiciones utilizan con frecuencia operadores de comparación, aunque no es la única forma de crearlas. Es posible utilizar cualquier objeto y evaluarlo en una función de los principios de evaluación booleana.

Un operador de comparación devuelve simplemente un valor booleano, que es Asimismo un objeto.

La instrucción *is* permite realizar una comparación sobre la identidad del objeto, y no sobre su valor. La instrucción *not* es, simplemente, una instrucción que interviene una condición, cuál sea.

**SELECTIVAS**

**IF**

La instrucción *if* es una palabra reservada el Python usada para ejecutar un bloque de sentencias si se cumple determinada condición representada por una expresión booleana.

En el caso del *if* simple se evalúa la expresión booleana y si el resultado es *True* se ejecuta el bloque de instrucciones asociado con él (su cuerpo) siguiéndose a continuación el flujo del programa. Si el resultado es *False* no ejecutaría el cuerpo del *if* y seguiría directamente con el flujo del programa.

Captura de pantalla de un celular

Descripción generada automáticamente

**IF - ELSE**

La instrucción combinada *if-else* añade al *if* la posibilidad de ejecutar un bloque de instrucciones cuando la condición es falsa.

Se evalúa la expresión lógica; si el resultado es *True* ejecuta un bloque de instrucciones y en caso contrario será otro bloque. La expresión booleana siempre es verdadero o falsa, por lo que está asegurada la ejecución de alguno de los dos bloques.

Una captura de pantalla de una red social

Descripción generada automáticamente

**IF – ELIF - ELSE**

Mediante *if-elif-else* podemos tener varias expresiones boolenas, que se evaluarán solamente si las anteriores han devuelto *Falso*, ya que en el momento en que alguna devuelve *True* se ejecuta el bloque de instrucciones asociado a ella y se sale de la instrucción combinada *if-elif-else*.

En cuanto una expresión booleana es verdadera, tras ejecutar las instrucciones asociadas a ella, se seguirá el flujo del programa. Si ninguna de ellas es verdadera, se ejecuta el bloque de instrucciones asociado al *else*.

Una captura de pantalla de una red social

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

**REPETITIVAS**

En nuestros programas necesitaremos ejecutar de forma reiterada (en bucle) bloques de código. Habra casos en los que sabremos el número exacto de repeticiones, y otros en los que conoceremos la condición de salida del bucle. Python nos proporcionan instrucciones para implementar bucles:

* ***For***, donde el bucle es controlado mediante la variable contador.
* ***While***, donde el bucle lo controla una condición lógica.

Con *for* tendremos, en los casos en que se pueda usar, un elemento más simple y cómodo, mientras que *while* nos permitirá más versatilidad.

**WHILE**

Mediante *while* entraremos en un bucle si se cumple una determinada condición lógica, y nos mantendremos en él mientras la citada condición sea verdadera.

La condición lógica de mantenimiento en el bucle es una expresión booleana que devolverá *True* o *False* computarse. En el primer caso de que resultara el bloque (lo que se denomina el cuerpo del *while*), tras lo cual volverá a comprobar la condición lógica, y así sucesivamente hasta que el resultado sea *False*, se salga del bucle *while,* y continúe el programa en la siguiente instrucción. Es importantísimo que en el cuerpo del *while* se modifiquen elementos que haga que la condición de mantenimiento en bucle sea falsa, dado que de lo contrario podríamos entrar en un bucle infinito, algo que “colgaría” nuestro programa.

Captura de pantalla de un celular

Descripción generada automáticamente Captura de pantalla de un celular

Descripción generada automáticamente

**FOR**

Mediante *for* creamos un bucle controlado por una variable contador que recorre la serie de valores que contiene una secuencia.

En ella se repite en ejecución el bloque de instrucciones (lo que se denomina el cuerpo del *for*) mientras el va pasando por los valores contenidos en la secuencia. Cuando se han recorrido todos, se termina el bucle.

Los 3 parámetros entrada para la primera sintaxis son:

* ***INICIO:*** Entero desde el cual comenzará la secuencia.
* ***FINAL:*** Entero posterior al que finalizara la secuencia, es decir, está llegará hasta final – 1
* ***PASO:*** Entero que determina el incremento (o decremento puede ser un número negativo) entre los valores de la secuencia.

**ITERACIÓN DE LISTAS**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**ITERACIÓN DE DICCIONARIOS**

**Captura de pantalla de un celular

Descripción generada automáticamente Captura de pantalla de un celular

Descripción generada automáticamente**

**BIBLIOTECAS**

Todas las funcionalidades de Python son proporcionadas a través de bibliotecas que se encuentran en la colección de The Python Standard Library, la mayoría de estas bibliotecas son multiplataforma. Las bibliotecas más usadas son:

* ***NumPy (Numerical Phyton):*** Es una de las bibliotecas más populares de Python, es usado para realizar operaciones con vectores o matrices de manera eficiente. Contiene funciones de álgebra lineal transformadas de Fourier, generación de números aleatorios e integración con Fortran, C y C++.
* ***SciPy (Numerical Phyton):*** Es una biblioteca que hace uso de Numpy y es utilizada para hacer operaciones más avanzadas como transformadas de Fourier, Algebra Lineal, Optimización, entre otras.
* ***Matplotlib:*** Es una biblioteca usada para generar una variedad de graficad en 2D y 3D, done una de las configuraciones de la gráfica es programable. Se puede usar comando de Latex para agregar ecuaciones matemáticas a las gráficas.
* ***Scikit Learn (Machine Learning):*** Esta biblioteca basada en los anteriores y contiene algoritmos de aprendizaje de máquina, reconocimiento de patrones y estadísticas para realizar clasificación, regresión, clustering, entre otros.
* ***Pandas (Manipulación de datos):*** Esta biblioteca es utilizada para manipulación de datos, contiene estructuras de datos llamadas data frames que se asemejan a las hojas de cálculo y a los cuales se le puede aplicar una gran cantidad de funciones.

**Una captura de pantalla de una red social

Descripción generada automáticamente**

**Una captura de pantalla de una red social

Descripción generada automáticamente**

**NUMPY**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**CREAR VECTORES Y MATRICES**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**Captura de pantalla de un celular con texto

Descripción generada automáticamente**

La diferencia entre shape y size es que shape indica el número de renglones y columnas que componen al vector o matriz. Mientras size indica el total de elementos.

**Captura de pantalla de un celular

Descripción generada automáticamente**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**ACCEDER A LOS ELEMNTOS DE LOS VECTORES Y MATRICES**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**RENGLONES, COLUMNAS Y ELEMENTOS**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**CONJUNTOS DE RENGLONES**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**SUBCONJUNTOS**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**CICLOS**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**GUARDAR Y CARGAR VARIABLES**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**OPERACIONES CON MATRICES**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**MATPLIOTLIB**

**Captura de pantalla de un celular

Descripción generada automáticamente**

**Una captura de pantalla de una computadora

Descripción generada automáticamente**

**Una captura de pantalla de una red social

Descripción generada automáticamente Una captura de pantalla de una red social

Descripción generada automáticamente**

**GRAFICACION**

***Matplotlib*** una biblioteca usada para generar gráficas en 2D y 3D, donde cada una de las configuraciones de la gráfica es programable. En el siguiente ejemplo se mostrará la configuración básica de una gráfica.

**Una captura de pantalla de una red social

Descripción generada automáticamente**

**Una captura de pantalla de una computadora

Descripción generada automáticamente**

**EJECUCIÓN DESDE VENTANAS DE COMANDOS**

Todo el código que se ha visto hasta el momento puede ser guardado en archivos de texto plano con la extensión ‘.py’. Para ejecutarlo desde la ventana de comandos se escribe el comando:

***python nombre\_archivo.py***

**ENTRADA DE DATOS**

Al igual que en otros lenguajes, también se puede se le puede pedir al usuario que introduzca ciertos datos de entrada cuando se ejecute un programa. Esto no se puede hacer desde la notebook, ya que los datos se introducen en las celdas que se van agregando a lo largo de la página, tal y como se ha venido manejando hast ahora. Como ejemplo se va a ejecutar el archivo lectura\_datos.py desde una ventana de comandos.

***python lectura\_datos.py***

**CONCLUSIONES**

Con esta práctica he podido observar que Python es un lenguaje de programación extremadamente versátil y con muchas funciones que se vuelven muy prácticas al escribir el código e igualmente en su ejecución. Además, he tenido la oportunidad de investigar que cuenta con más de 300 módulos de biblioteca estándar que contienen módulos y clases para una amplia variedad de tareas d6e programación.

Incluso pienso que cuando uno ha dominado Python uno puede acostumbrarse tanto a sus características, particularmente su modelo dinámico de enlace en tiempo de ejecución y sus muchas bibliotecas, que puede ser difícil de aprender y sentirse cómodo en otros lenguajes de programación. Específicamente, la necesidad de declarar valores de variable "tipos" y "cast" de un tipo a otro y los requisitos sintácticos para agregar punto y coma y llaves utilizados por otros lenguajes de programación pueden ser vistos como tediosos.

**BIBLIOGRAFÍA**

* Cuevas, A. (2019). *Programar con Python 3* (1ª edición). Madrid, España: RA-MA.
* Guagliano, C. (2019). *Programación en Python 1: Entorno de Programación- Sintaxis. Estructuras de Control* (1ª edición). Buenos Aires, Argentina: Six Ediciones.