Programación II

Práctica 1

Tecnicatura en Análisis de Sistemas - Instituto Superior de Formación Docente y Técnica 210

Temas: Variables, Strings, Listas, Diccionarios, Tuplas, Selección, Condicionales, Excepciones, Modularidad, Funciones, Iteración, TAD, POO, Archivos, Networking, Sockets, Puertos, Seguridad en Redes.

1. Repaso

a. **Variables**: Cree una variable llamada **port** y asigne el entero **22**, luego cree una variable llamada **banner** y asigne la cadena **“SSH Port”.** Imprima en pantalla la siguiente frase: “{+] Checking for **VARIABLE_BANNER** on port **VARIABLE_PORT**-“

b. **Strings**: Lea la documentación oficial de Python sobre cadenas y realice las siguientes operaciones dando ejemplos de cómo las realiza:

   i. Dada una cadena en minúsculas pasarla a mayúsculas.

   ii. Dada una cadena en mayúsculas pasarla a minúsculas.

   iii. Reemplace una palabra de una cadena por otra.

   iv. Reporte el offset (ubicación) de una palabra específica en una cadena.

c. **Listas**: Lea la documentación oficial de Python sobre listas y realice las siguientes operaciones dando ejemplos de cómo las realiza:

   i. Cree una **lista** vacía

   ii. **Agregue** a la lista al **final** los siguientes enteros: **21, 80, 443, 25, 22, 8080**

   iii. **Imprima** en pantalla la lista con los elementos agregados.

   iv. Imprima en pantalla la lista **ordenada**.

v. Imprima en pantalla la **posición** en la lista del entero **80**.

vi. **Borre** de la lista el entero **443** e imprima la **longitud** de la lista.

vii. **Inserte** al **inicio** de la lista el entero 8100.

d. **Diccionarios:** Lea la documentación oficial de Python sobre diccionarios y realice las siguientes operaciones dando ejemplos de cómo las realiza:

i. Investigue a qué servicios de red comunes corresponden los enteros en la lista realizada en el punto anterior.

ii. Cree un **diccionario** llamado servicios y asigne en cada **clave** el nombre del servicio y en cada **valor** su puerto número de puerto correspondiente (**ej: {'ssh': 22 }**).

iii. Imprima en pantalla sólo las **claves** del diccionario.

iv. Imprima en pantalla **cada ítem** del diccionario pero en una **lista** de **tuplas clave valor**.

v. Imprima en pantalla si existe la clave '**ftp**' en el diccionario y si existe imprima su **valor**.

vi. Imprima en pantalla para cada elemento del diccionario la siguiente frase: "[+] Found vulnerabilities with **NOMBRE_SERVICIO** on port **NOMBRE_PUERTO** "

e. **Condicionales / Selección:**

i. Desarrolle un programa que dado un número ingresado por el usuario lo divida por dos e imprima si su resultado es par o impar.

ii. Desarrolle un programa que dada una secuencia de número ingresados por el usuario los divida por dos e imprima si es par o impar. ¿Qué ocurre si el usuario ingresa un 0 o un carácter? ¿Cómo lo evitaría?

f. Excepciones: Lea la documentación oficial de Python sobre errores y excepciones y realice las siguientes operaciones dando ejemplos de cómo las realiza:

i. Logre un **error** de división por cero.

ii. Intente **escapar** al error de división por cero con una **excepción general**.

iii. Logre en un sólo programa un **error de división por cero,** un **error por valor** y un **error de índice fuera de rango.**

iv. Intente **escapar** a los errores anteriores con excepciones general e **imprima la excepción generada.**

v. Intente **escapar** a los errores anteriores con **excepciones específicas** para cada error e **imprima la excepción generada.** Para cada excepción atrapada gestione su solución.

vi. A partir de éste punto **desarrolle** sus programas utilizando **excepciones** según cada caso (de ser necesario).

g. **Funciones:** Lea la documentación oficial de Python sobre control de flujo y responda las siguientes preguntas dando ejemplos con código:

i. ¿Qué son y para qué sirven las **variables por defecto**?

ii. ¿Qué son y para qué sirven **los argumentos clave (keyword arguments o kwargs)**?

iii. Realice una función llamada **connect** que acepte dos parámetros: **ip** y **port.** La función deberá imprimir en pantalla "[+] Trying to connect to **VARIABLE_IP** for port **VARIABLE_PORT** " y retornar una tupla con primer valor **VARIABLE_IP** y segundo valor **VARIABLE_PORT.**

iv. Realice una función llamada **checkvulns** que acepte dos parámetros: **bannerlist** y **banner** (la variable **bannerlist** tiene por defecto asignada la variable **bannerfile** que deberá ser declarada como string vacío al principio del programa fuera de la función). La función deberá buscar la cadena en la variable **banner** dentro de la lista de cadenas **bannerlist** y en caso de encontrarla deberá imprimir en pantalla "[+] The Service **VARIABLE_BANNER** seems to be vulnerable!" y retornar la cadea en la variable **banner** y en caso contrario deberá imprimir "[-] The Service **VARIABLE_BANNER** seems to not be vulnerable." y retornar **False.**

v. A partir de éste punto **desarrolle** sus programas utilizando **funciones** con **variables por defecto** y **kwargs** según cada caso (de ser necesario).

h. **Iteración:**

i. Imprima en pantalla una lista de todas las direcciones **IP** dentro de su rango de red local (**ej: 192.168.0.1 ... 192.168.0.254**).

ii. Dado el **diccionario de servicios y puertos** creado en el punto **d** imprima en pantalla todos los números de los puertos.

iii. Desarrolle una función que dada una lista de puertos imprima en pantalla para cada puerto y para cada ip en su rango de red: "[+] Checking **VARIABLE_IP**:**VARIABLE_PUERTO**

2. Modularidad

a. ¿Qué es la modularidad? Describa brevemente y cite al menos dos ejemplos de la vida cotidiana y al menos dos relacionada con la informática.

b. Describa brevemente los conceptos de acoplamiento y cohesión que la modularidad debe seguir. Dé ejemplos con Python 3.

3. Archivos

a. Lea la documentación oficial de Python sobre escribir y leer archivos y realice las siguientes operaciones dando ejemplos de cómo las realiza:

i. Descargue el diccionario de passwords en un directorio donde esté trabajando. Desarrolle un programa en python que lea línea por línea del diccionario de passwords y para cada línea imprima en pantalla: "[+] Trying password: **PASSWORD**"

ii. Desarrolle una función que dadas dos palabras pasadas por parámetro copare si las palabras tienen la misma cantidad de vocales en total. Si las tiene, escriba dichas palabras en otro archivo.

iii. Caso de Estudio: ZIP PASSWORD BRUTE FORCER.
Lea la documentación oficial de Python sobre la librería ZipFile y desarrolle:

1. Una función que dado una ubicación de un archivo zip y un password pasados como parámetro intente extraer el contenido

utilizando dicha password y si lo logra imprima en pantalla “[+] Password found **PASSWORD** “ y escriba en un nuevo archivo la password. En caso contrario la función no debe arrojar errores (recordar **excepciones**) y debe continuar la ejecución normal del programa.

2. Una función que dada una ubicación de un archivo de passwords pasada como parámetro lea cada línea y llame con cada línea a la función desarrollado anteriormente. Cuando la función inicia debe imprimir en pantalla “[+] Starting brute force with password file **PASSWORDFILE** for zipfiiile **ZIPFILE**”.

4. Tipo Abstracto de Dato

a. Explique brevemente qué es un Tipo Abstracto de Datos (TAD o ADT). Explique con ejemplos cómo lo podría utilizar en un programa o en una solución a un problema.

b. Explique con sus palabras cómo relaciona el concepto de Tipo Abstracto de Datos y la Modularidad.

5. Programación Orientada a Objetos

a. Explique brevemente qué es la **Programación Orientada a Objetos** (POO o OOP). Explique con ejemplos cómo la podría utilizar en un programa o en una solución a un problema.

b. Explique con sus palabras la relación entre la **Modularidad** y la **Programación Orientada a Objetos**.

c. Explique con sus palabras la relación entre el **Tipo Abstracto de Datos** y la **Programación Orientada a Objetos.**

d. Lea la documentación oficial de Python sobre clases y desarrolle:

   i. Cree una clase **User** con:

      1. **Atributos**:

a. **fullname**: String completo que representa nombre y apellido separados por espacios, puede tener varios nombres y varios apellidos.

b. **date_of_birth**: String que indica la fecha de nacimiento respetando el formato: YYYYMMDD (Ej: 19890216).

2. **Métodos**:

a. **__init__**: Inicializa a la instancia del objeto al ser creado (debe inicializar sus atributos con los parámetros pasados en la creación).

b. **salutte**: imprime en pantalla el texto :"Hello! I'm **FULLNAME**. Nice to meet you!.

c. **age**: retorna la edad del objeto calculada a partir de date_of_birth (utilizar datetime).

d. **name**: retorna el nombre del objeto, sin el apellido.

e. **lastname**: retorna el apellido el objeto, sin el nombre.

ii. Investigue el comando help(**instancia de un objeto**) Ej: help(user1).

iii. Instancie 3 objetos **User** y agreguelos en una **lista**. Desarrolle un programa que **recorra** la lista de objetos user y para cada uno **llame** a cada uno de los **métodos** desarrollados en la clase e imprima en pantalla sus resultados.

e. Desarrolle la clase **Car**:

i. **Atributos**:

1. **gas**: entero que representa el combustible del automóvil.

ii. **Métodos**:

1. **start**: imprime en pantalla "Car started!" si la variable interna gas es mayor a 0, si no imprime en pantalla "No Gas, cannot start the car."

2. **drive**: resta en 1 la variable interna gas e imprime "Car has **GAS** left", si la variable gast es menor a 1 no resta e imprime: "No Gas, cannot drive the car."

f. Desarrolle la clase **Nodo**:

   i. **Atributos**:

      1. **dato**: la variable dato puede ser de cualquier tipo y representa un dato interno al objeto nodo en sí mismo.
      2. **siguiente**: la variable siguiente es de tipo Nodo y representa al objeto Nodo siguiente.

   ii. **Métodos**:

      1. **last**: retorna el objeto Nodo enlazado que no tenga siguiente.
      2. **first**: retorna el objeto Nodo enlazado que no esté enlazado por ningún otro objeto Nodo.
      3. **len**: retorna la longitud de todos los objetos Nodos enlazados.
      4. **alldata**: retorna una lista con todos los dato de todos los Nodos enlazados.

BONUS!

Networking con Python:

1. Investigue qué es un **socket de red** y explique brevemente.
2. Investigue la librería **socket** de Python y dé al menos ejemplos de la utilización de 3 funciones principales de la misma.
3. Desarrolle un programa que intente conectarse a una **ip** de alguna máquina existente (puede ser en una **red local** o en internet a través de **Shodan**) en su puerto **21** e imprima la respuesta de dicho intento de conexión.