1- ¿Por qué decimos que Python es "interpretado"? ¿Cual es la diferencia con un lenguaje "compilado"?

Porque no compila en lenguaje maquina(binario) sino que se mantiene, comparte y distribuye en el lenguaje de tipo scripting directamente(texto plano). Y en tiempo de ejecución hay un intérprete que traduce estas instrucciones del script de python en lenguaje máquina y finalmente lo ejecuta.

2- ¿Que queremos decir cuando decimos que Python tiene "tipado dinámico"?

Se puede cambiar de tipo a una variable sin ningun problema. (en C no se puede por ejemplo)

3- ¿Que queremos decir cuando decimos que Python tiene "garbage collector"?
No hay que liberar memoria

4- ¿Por qué la indentanción es importante en Python?
Ya que no usa llaves{}

5- ¿Cual es la diferencia entre tupla, lista y diccionario?
Las tres son estructuras de datos, solo que tienen diferentes implementaciones, operaciones y sirven para distintos propósitos.
Las listas son modificables
Las tuplas NO
Los diccionarios no tienen indice, hay que usar la clave!

En Python, las tuplas, listas y diccionarios son estructuras de datos diferentes que se utilizan para almacenar y manipular datos de diferentes maneras.

Una tupla es una colección inmutable y ordenada de elementos, separados por comas y encerrados entre paréntesis. Una vez que se crea una tupla, no se puede modificar su contenido, lo que significa que no se pueden agregar, eliminar o modificar elementos. Las tuplas se utilizan generalmente para almacenar datos que no cambian a lo largo del tiempo, como las coordenadas de un punto en un plano cartesiano.

Una lista, por otro lado, es una colección mutable y ordenada de elementos, separados por comas y encerrados entre corchetes. A diferencia de las tuplas, las listas pueden ser modificadas después de su creación. Se pueden agregar, eliminar y modificar elementos en una lista. Las listas se utilizan comúnmente para almacenar datos que pueden cambiar a lo largo del tiempo, como los nombres de los estudiantes en una lista de asistencia.

Por último, un diccionario es una colección mutable y desordenada de pares clave-valor, encerrados entre llaves. Cada elemento del diccionario está formado por una clave y su valor correspondiente. Los diccionarios se utilizan para almacenar y acceder a datos de forma eficiente, ya que se pueden buscar valores utilizando su clave en lugar de tener que recorrer la lista completa de elementos. Por ejemplo, un diccionario puede almacenar la información de contacto de un conjunto de personas, donde la clave podría ser el nombre

de la persona y el valor podría ser su dirección de correo electrónico o su número de teléfono.

En resumen, las tuplas son estructuras de datos inmutables, las listas son estructuras de datos mutables y los diccionarios son estructuras de datos desordenadas y mutables que se utilizan para almacenar y manipular datos de diferentes maneras en Python.

6- ¿Qué quiere decir variables "inmutables"? ¿Cuales son estas variables? ¿Cuales no lo son?

Inmutable: tuplas, numero y string (si asignamos otra variable, pasan a ser dos objetos independientes)
Mutables: listas, diccionarios y clases (si asignamos otra variables apunta al mismo objeto), en estos casos si se quiere duplicar hay que hacer una copia explicitamente.

7- ¿Por qué es importante usar estructuras de try/except en aplicaciones de redes?
Las excepciones son errores detectados en ejecución y no son incondicionalmente fatales. Es importante el uso de estructuras try/except en aplicaciones de redes ya que por este medio podemos seguir gestionando la ejecución de un programa si alguna función/método de la API que utilicemos para el manejo de redes falla.

8- ¿Qué es PEP8?
Una convención de escritura que está destinada a hacer más legible el código.

9- ¿Cuántos espacios recomienda PEP8 para un nivel de indentación?
4

10- ¿Cuál es la ventaja de debuggear con depuradores en lugar de print?
○ Más fácil de controlar en bucles
○ Cualquier variable
○ Nos olvidamos de los print que generan mal aspecto
○ Puntos de interrupción en el código para revisar valor de variables
○ Estudio del código línea por línea, función por función, y estudiar la evolución de las variables

11- ¿Qué es un protocolo? ¿Puede dar algunos ejemplos?
Un protocolo define el formato y el orden de los mensajes intercambiados entre dos o más entidades que se comunican, así como las acciones tomadas al producirse la transmisión y/o recepción de un mensaje u otro suceso.
Protocolos de transporte, TCP y UDP
Protocolos de enlace, Ethernet y Wifi

Un protocolo de comunicación es un conjunto de reglas, conjunto de formatos, una definición, una especificación de cómo se tiene que transmitir información y cuales son las

acciones que se espera que cada uno de los nodos de esta red tenga que tomar ante diferentes situaciones.

12- ¿Qué es un socket? ¿Para qué sirve?
Un proceso es análogo a una casa y un socket es análogo a la puerta de la casa.
Un socket es la interfaz entre la capa de aplicación y la de transporte
Sirve para comunicarse entre procesos.

13- ¿Cómo funciona el paradigma cliente/servidor? ¿Cómo se ve esto en la programación con sockets?
En el contexto de una sesión de comunicación entre una pareja de procesos, el proceso que inicia la comunicación (es decir, que inicialmente se pone en contacto con el otro proceso al principio de la sesión) se designa como el cliente. El proceso que espera a ser contactado para comenzar la sesión es el servidor.

El servidor crea un socket y espera a que el cliente se conecte a él. Una vez que el cliente se ha conectado, el servidor acepta la conexión y se establece una conexión de comunicación bidireccional entre el cliente y el servidor.

El cliente envía solicitudes al servidor a través del socket, y el servidor procesa estas solicitudes y envía una respuesta de vuelta al cliente a través del mismo socket. El cliente y el servidor pueden enviar y recibir datos a través del socket en cualquier momento mientras la conexión está activa.

14- ¿Qué datos necesito para identificar unívocamente un servidor en Internet? ¿Cómo se expresa esto en Python?
Para identificar un servidor en Internet de manera unívoca se necesitan dos tipos de datos: la dirección IP del servidor y el número de puerto que se está utilizando para el servicio que se desea acceder.

15- ¿Por qué a veces usamos URLs en lugar de IPs para identificar hosts? ¿Cómo se traducen estos nombres a IPs?
A veces usamos URLs en vez de IPs porque es más fácil para los humanos recordar un string (por así decirlo) a un número de 32 bits, también porque podemos saber el URL y no la dirección IP asociada a este. Existe una relación inyectiva entre URLs e IPs, entonces podemos saber la URL y “traducirla” mediante el servicio DNL.
Estas URLs se pueden traducir a IPs a través de una función que nos provee sockets que es gethostbyname(), en donde se le pasa como parámetro el URL y devuelve su dirección IP.

16- ¿Por qué existen puertos reservados o de "servicios conocidos"? ¿Puede nombrar algunos?

Al reservar puertos específicos para servicios conocidos, los programadores pueden garantizar que los programas en la red se comuniquen correctamente y eviten conflictos de puertos. Esto ayuda a garantizar la estabilidad y seguridad de las comunicaciones de red.

- 21 FTP
- 22 SSH
- 23 Telnet
- 25 SMTP (Mail)
- 80 HTTP (Web)
- 110 POP3 (Mail)
- 443 HTTPS (web)

17- ¿Para qué se usan los puertos no registrados?
Los puertos no registrados se utilizan para aplicaciones personalizadas o específicas que no están asociadas con ningún servicio o aplicación estándar.
Los programadores pueden tener mayor flexibilidad para crear aplicaciones y servicios personalizados que puedan comunicarse a través de la red

18- ¿Cuál es la diferencia entre Stream (TCP) y Datagram (UDP), desde la perspectiva del socket?
TCP incluye un servicio orientado a la conexión y un servicio de transferencia de datos fiable (entregar todos los datos sin errores y en el orden correcto)
UDP es un protocolo de transporte ligero y simple que proporciona unos servicios mínimos. No está orientado a la conexión, por lo que no tiene lugar un procedimiento de negociación antes de que los dos procesos comiencen a comunicarse. UDP proporciona un servicio de transferencia de datos no fiable; es decir, cuando un proceso envía un mensaje a un socket UDP, el protocolo UDP no ofrece ninguna garantía de que el mensaje vaya a llegar al proceso receptor. Además, los mensajes que sí llegan al proceso receptor pueden hacerlo de manera desordenada.

19- ¿Qué es el protocolo HTTP? ¿Para qué sirve?
Es el protocolo de la capa de aplicación de la Web.
HTTP se implementa mediante dos programas: un programa cliente y un programa servidor. Ambos programas, que se ejecutan en sistemas terminales diferentes, se comunican entre sí intercambiando mensajes HTTP. HTTP define la estructura de estos mensajes y cómo el cliente y el servidor intercambian los mensajes.
La mayoría de las páginas web están constituidas por un archivo base HTML y varios objetos referenciados.
HTTP define cómo los clientes web solicitan páginas web a los servidores y cómo estos servidores web transfieren esas páginas a los clientes.

20- ¿Cuál es la diferencia entre HTTP y HTML?
HTML es un estándar para los formatos de documentos web.
HTTP es el protocolo de la web.

HTML, por otro lado, es un lenguaje de marcado utilizado para crear páginas web. Define la estructura y el contenido de una página web utilizando etiquetas y atributos que se

interpretan para renderizar la página web en un navegador. HTML se utiliza para crear el contenido que se entrega a través del protocolo HTTP.
En resumen, HTTP es el protocolo utilizado para enviar y recibir datos a través de Internet, mientras que HTML es un lenguaje utilizado para crear el contenido que se entrega a través del protocolo HTTP.

21- La instrucción socket.send(), ¿Envía siempre todos los datos? ¿Qué retorna dicho método?
No siempre, es posible que algunos bytes no se hayan enviado debido a una interrupción de la conexión de red o un error en la transmisión.
Retorna el número de bytes enviados correctamente

22- ¿Para qué sirve el método socket.sendall()? ¿En qué cambia respecto a socket.send()?
Este método continúa enviando datos de bytes hasta que se hayan enviado todos los datos o se produzca un error. Devuelve None en caso de éxito. En caso de error, se genera una excepción y no hay forma de determinar cómo muchos datos, si los hubo, se enviaron con éxito.
En resumen, la función send envía una cantidad específica de datos a través del socket y devuelve el número de bytes enviados, mientras que la función sendall envía todos los datos a través del socket y lanza una excepción si no se pueden enviar todos los datos.

23- La instrucción socket.recv(), ¿Recibe siempre todos los datos enviados por el host transmisor?
No, la función recv en la programación de sockets no siempre recibe toda la información de una sola vez. La cantidad de datos que se reciben en una llamada a la función recv puede ser menor que la cantidad de datos que se enviaron originalmente.

24- ¿En qué casos socket.recv() no recibiría todos los datos? ¿Qué toma como argumento?
La cantidad de datos que se reciben en una llamada a recv depende de varios factores, como el tamaño del búfer de recepción, la cantidad de datos disponibles en el búfer de recepción en ese momento y la configuración de la conexión de socket.
Por lo tanto, es posible que deba llamar a la función recv varias veces para recibir todos los datos que se enviaron a través del socket. Puede utilizar un bucle para recibir los datos en varias llamadas a recv hasta que se hayan recibido todos los datos.
El argumento que le pasamos es el tamaño del búfer de recepción, que es la cantidad máxima de datos que se pueden recibir en una sola llamada a la función recv. Si los datos que se están recibiendo son más grandes que el tamaño del búfer, entonces se pueden recibir en múltiples llamadas a la función recv.

25- ¿Qué devuelve el método socket.accept()?
connectionSocket, addr = socket.accept()
Cuando un cliente llama a esta puerta, el programa invoca el método accept() para el serverSocket, el cual crea un nuevo socket en el servidor, denominado connectionSocket, dedicado a este cliente concreto. Y en addr guarda la ip y puerto del cliente.

26- ¿Qué argumento toma el método socket.listen()? ¿Qué querría decir entonces socket.listen(5)?
El método socket.listen() es utilizado en Python para indicar al socket que empiece a escuchar conexiones entrantes. Este método toma como argumento el número máximo de conexiones en cola que el socket puede manejar.

27- ¿Qué es base64? ¿Para qué la usamos en el laboratorio?
La codificación Base64 es el proceso de convertir los datos binarios, en un juego de caracteres limitado a 64 caracteres. Como hemos dicho en la primera sección, esos caracteres son A-Z, a-z, 0-9, +, y / (¿Los has contado?, ¿te has dado cuenta de que se suman 64?). Este conjunto de caracteres se considera el conjunto de caracteres más común, y se conoce como el Base64 de MIME. Utiliza A-Z, a-z, 0-9, +, y / para los primeros 62 valores, y +, y / para los dos últimos valores.

Base64 es un método de codificación aplicado a datos binarios para representar la información obtenida en una cadena de caracteres en código ASCII. Esto es, la información binaria se lee en grupos de bits y cada grupo se traduce en su representación ASCII-Base64.
Los archivos con datos binarios, bytes que representan información que no es de texto, como imágenes, pueden corromperse fácilmente cuando se transfieren y procesan a sistemas de solo texto. Entonces, usamos la codificación Base64 que nos permitirá convertir bytes que contienen datos binarios o de texto en caracteres ASCII, mejorando las posibilidades de que varios sistemas los procesen correctamente.

28- ¿Qué pasa si queremos enviar un archivo contiene los caracteres \r\n? ¿Cómo lo soluciona su código?

29- ¿Por qué es mayor el tamaño de los datos codificados en base64 que el dato original?
Los datos codificados en Base64 terminan siendo más grandes que los datos originales, de manera que, como hemos dicho antes, por cada 3 bytes de datos binarios, hay al menos 4 bytes de datos codificados en Base64. Esto es debido al hecho de que estás apretando los datos en un conjunto más pequeño de caracteres.
tamaño original *(4/3)

30- ¿Qué estrategias se pueden considerar para un servidor multi-cliente? ¿Podría comentar sobre alguna?
Forks, Trheads, Async