Tema:

Comparativa de Lenguajes de Programación emergentes para aplicaciones en Infraestructura

Integrantes: Martin Zambonino, Enrique Pérez.

Pregunta 1

Enunciado:

¿Cuál es el objetivo principal del lenguaje Rust y cómo garantiza la seguridad en la programación de sistemas?

Opciones de respuesta:

a) Ofrecer una interfaz gráfica intuitiva y garantizar la seguridad mediante el uso de contraseñas encriptadas.
b) Proporcionar una sintaxis simplificada y garantizar la seguridad mediante la verificación en tiempo de ejecución.
c) Ofrecer un alto rendimiento y garantizar la seguridad mediante la propiedad y el préstamo.
d) Facilitar la interoperabilidad con otros lenguajes y garantizar la seguridad mediante el uso de recolectores de basura.

Respuesta Correcta: c. Ofrecer un alto rendimiento y garantizar la seguridad mediante la propiedad y el préstamo.

Justificación de la opción correcta: Rust garantiza la seguridad mediante los conceptos de propiedad y préstamo, que son verificados por el compilador en tiempo de compilación, evitando problemas como fugas de memoria o uso indebido, sin necesidad de comprobaciones en tiempo de ejecución.

Justificación de las opciones incorrectas:

a. Rust no se enfoca en interfaces gráficas ni utiliza contraseñas encriptadas para garantizar la seguridad.
b. Aunque la verificación en tiempo de compilación es importante, no es la característica distintiva de Rust en términos de seguridad.
d. Rust evita el uso de recolectores de basura para garantizar la seguridad, prefiriendo sus propios mecanismos de gestión de memoria.

Pregunta 2

Enunciado:

¿Qué características hacen de Python un lenguaje adecuado para aplicaciones de infraestructura y qué ventajas ofrece en términos de desarrollo?

Opciones de respuesta:

a) Sintaxis complicada, baja flexibilidad, y una pequeña comunidad.
b) Sintaxis simple, flexibilidad, y una amplia comunidad que proporciona documentación y soporte.
c) Alto rendimiento, rigidez en la sintaxis, y escasez de librerías.
d) Dificultad en la lectura de código, baja portabilidad, y comunidad limitada.

Respuesta Correcta: b. Sintaxis simple, flexibilidad, y una amplia comunidad que proporciona documentación y soporte.

Justificación de la opción correcta: Python es adecuado para aplicaciones de infraestructura debido a su sintaxis simple y clara, su flexibilidad, y una amplia comunidad que proporciona documentación, soporte, librerías y herramientas, lo que reduce el tiempo de desarrollo y errores.

Justificación de las opciones incorrectas:

a. Python es conocido por su sintaxis simple y una gran comunidad, no por ser complicado ni tener baja flexibilidad.

c. Aunque Python es flexible, no es conocido por tener alto rendimiento en comparación con lenguajes compilados.

d. Python es fácil de leer, altamente portable y tiene una comunidad grande y activa.

Pregunta 3

Enunciado:

¿Cómo ha evolucionado Python desde su creación y cuáles son algunas de las funcionalidades clave que ha incorporado a lo largo de los años?

Opciones de respuesta:

a. Se ha mantenido sin cambios significativos y no ha incorporado nuevas funcionalidades.
b. Ha reducido su flexibilidad y eliminado el soporte para programación asíncrona.
c. Ha evolucionado significativamente, incorporando pip, Django, NumPy y soporte para programación asíncrona.
d. Se ha centrado en eliminar el tipado dinámico y mejorar la interoperabilidad con COBOL.

Respuesta Correcta: c. Ha evolucionado significativamente, incorporando pip, Django, NumPy y soporte para programación asíncrona.

Justificación de la opción correcta: Python ha incorporado numerosas funcionalidades a lo largo de los años, como el sistema de gestión de paquetes pip, el framework web Django, la librería científica NumPy, y el soporte para programación asíncrona, reflejando su evolución y crecimiento en la comunidad.

Justificación de las opciones incorrectas:

a. Python ha tenido cambios significativos y ha incorporado muchas nuevas funcionalidades desde su creación.
b. Python ha aumentado su flexibilidad y añadido soporte para programación asíncrona, no lo contrario.
d. Python mantiene su tipado dinámico y no se ha enfocado en mejorar la interoperabilidad con COBOL.

Pregunta 4

Enunciado:

¿Por qué se considera ampliamente a Python para desarrollar aplicaciones de infraestructura, a pesar de sus desventajas en sobrecarga y latencia en comparación con lenguajes compilados?

Opciones de respuesta:

a. Python permite la integración con bibliotecas y módulos en lenguajes compilados como C y C++, lo cual ayuda a mitigar los problemas de rendimiento en las secciones críticas del código.
b. Python ha implementado recientemente un sistema de tipado estático completo que elimina los problemas de sobrecarga y latencia asociados a su tipado dinámico.
c. Las mejoras en el recolector de basura de Python, específicamente con el uso de técnicas como la recopilación generacional, han eliminado las desventajas de sobrecarga y latencia.
d. El desarrollo de frameworks y herramientas avanzadas ha permitido que Python maneje eficientemente las operaciones de I/O y las cargas de trabajo paralelas, reduciendo significativamente la latencia.

Respuesta correcta:

a. Python permite la integración con bibliotecas y módulos en lenguajes compilados como C y C++, lo cual ayuda a mitigar los problemas de rendimiento en las secciones críticas del código.

Justificación de la respuesta correcta:

Esta es la respuesta correcta porque la integración de Python con bibliotecas y módulos en lenguajes compilados (por ejemplo, mediante el uso de Cython, ctypes o PyBind11) permite que las partes críticas del código se ejecuten con la eficiencia de los lenguajes compilados, mejorando así el rendimiento general.

Justificación de las respuestas incorrectas:

b. Aunque Python ha introducido type hints y el módulo mypy para proporcionar tipado estático opcional, no es un sistema de tipado estático completo como el de los lenguajes compilados. Estos tipos se utilizan principalmente para mejorar la legibilidad y la detección de errores en tiempo de desarrollo, pero no afectan la ejecución del código en tiempo de ejecución.
c. Aunque el recolector de basura de Python ha mejorado con técnicas como la recolección generacional, no ha eliminado por completo la sobrecarga y latencia. Estas técnicas mejoran la gestión de memoria, pero Python aún enfrenta problemas de rendimiento en comparación con lenguajes que no dependen de un recolector de basura.
d. Si bien existen frameworks como asyncio, Twisted, y herramientas como multiprocessing y concurrent.futures que mejoran la gestión de I/O y permiten la paralelización, estas no eliminan completamente la latencia y sobrecarga inherente al intérprete de Python y su Global Interpreter Lock (GIL).

Pregunta 5

Enunciado

¿Por qué el lenguaje Rust es altamente eficiente en el desarrollo de aplicaciones de infraestructura comparado con Python?

Opciones de respuesta:

a. Rust es un lenguaje compilado que genera código nativo altamente optimizado, proporcionando un control de bajo nivel sobre los recursos del sistema y una gestión eficiente de la memoria sin necesidad de un recolector de basura.
b. Rust tiene un recolector de basura más avanzado que Python, que minimiza la latencia y sobrecarga durante la ejecución del programa.
c. Rust permite una mayor concurrencia y paralelismo al no estar limitado por un Global Interpreter Lock (GIL) como en Python, lo que resulta en un mejor rendimiento en aplicaciones multicore.
d. Rust es interpretado, lo que le permite realizar optimizaciones en tiempo de ejecución y ajustar su rendimiento según las necesidades específicas de la aplicación.

Respuesta correcta:

a. Rust es un lenguaje compilado que genera código nativo altamente optimizado, proporcionando un control de bajo nivel sobre los recursos del sistema y una gestión eficiente de la memoria sin necesidad de un recolector de basura.

Justificación de la respuesta correcta:

Rust es un lenguaje compilado que genera código nativo optimizado, lo cual permite una gestión de recursos eficiente y un control de bajo nivel que es esencial para el desarrollo de aplicaciones de infraestructura. La ausencia de un recolector de basura también contribuye a la eficiencia y predictibilidad del rendimiento.

Justificación de las respuestas incorrectas:

a. Rust no tiene un recolector de basura en absoluto. En cambio, maneja la memoria a través de un sistema de propiedad y préstamos que elimina la necesidad de un recolector de basura, lo que reduce la latencia y sobrecarga asociadas a la gestión automática de memoria en Python.
b. Rust no tiene un GIL, lo que permite una verdadera concurrencia y paralelismo. Python, en particular CPython, está limitado por el GIL, lo que restringe el rendimiento en aplicaciones multihilo. Rust permite aprovechar completamente los sistemas multicore, lo que es crucial para aplicaciones de infraestructura de alto rendimiento.
c. Rust es un lenguaje compilado, no interpretado. Su compilación a código nativo permite optimizaciones en tiempo de compilación que resultan en un rendimiento altamente eficiente en comparación con los lenguajes interpretados como Python. La naturaleza interpretada de Python introduce una sobrecarga en tiempo de ejecución que Rust no tiene.

Pregunta 6

Enunciado

¿Por qué muchos desarrolladores eligieron Python para crear frameworks y bibliotecas con el enfoque de crear aplicaciones de infraestructura?

Opciones de respuesta:

a. Python es un lenguaje compilado que genera código nativo altamente optimizado, lo que lo hace ideal para aplicaciones de infraestructura.

b. Python ofrece una sintaxis clara y concisa que facilita el desarrollo rápido y la comprensión del código, lo que permite a los desarrolladores centrarse en la lógica de negocio en lugar de los detalles de implementación.

c. Python permite implementar un recolector de basura en tiempo de compilación que elimina la sobrecarga de la gestión de memoria en tiempo de ejecución.

d. Con Python se puede implementar un avanzado sistema de tipado estático que mejora la seguridad y el rendimiento del código en aplicaciones de infraestructura.

Respuesta correcta:

b. Python ofrece una sintaxis clara y concisa que facilita el desarrollo rápido y la comprensión del código, lo que permite a los desarrolladores centrarse en la lógica de negocio en lugar de los detalles de implementación.

Justificación de la respuesta correcta:

Esta es la respuesta correcta porque la sintaxis simple y legible de Python permite un desarrollo más rápido y reduce la barrera de entrada para los nuevos desarrolladores. Esto facilita la creación de frameworks y bibliotecas que pueden ser rápidamente adoptados y utilizados por la comunidad, acelerando el desarrollo de aplicaciones de infraestructura.

Justificación de las respuestas incorrectas:

a. Python es un lenguaje interpretado, no compilado. Aunque algunas optimizaciones pueden realizarse mediante el uso de bibliotecas escritas en C o C++, Python no genera código nativo altamente optimizado por sí mismo.
c. Python no tiene un recolector de basura en tiempo de compilación. La gestión de memoria en Python se realiza en tiempo de ejecución mediante un recolector de basura, lo que puede introducir sobrecarga y latencia en comparación con lenguajes que no dependen de esta técnica.
d. Aunque Python ha introducido type hints y herramientas como mypy para análisis de tipos estáticos, no es un sistema de tipado estático completo como en lenguajes como Rust o C++. Python sigue siendo un lenguaje de tipado dinámico, y estas herramientas se utilizan principalmente para mejorar la legibilidad y detección de errores en tiempo de desarrollo.