El lenguaje escogido para este exámen es:

Python

Python es un lenguaje de programación de alto nivel, interpretado, orientado a objetos y de propósito general. Fue creado por Guido van Rossum y lanzado por primera vez en 1991.

Python es conocido por su sintaxis clara y legible, lo que facilita el aprendizaje y la escritura de código. Su filosofía de diseño enfatiza la legibilidad del código y la simplicidad, utilizando palabras clave en inglés y una estructura de código limpia.

I. Creación y manipulación de objetos.

1. Clases y objetos.

Casi todo en Python es un objeto.

Una clase es una maqueta o prototipo definido por el usuario a partir del cual se crean los objetos. Las clases proporcionan un medio para agrupar datos (atributos) y funcionalidades (métodos).

Al crear una nueva clase, se crea un nuevo tipo de objeto, lo que permite crear nuevas instancias de ese tipo. Cada instancia de clase (objeto) puede tener atributos asociados para mantener su estado. Las instancias de clase también pueden tener métodos (definidos por su clase) para modificar su estado.

2. Constructores.

El constructor es un método especial que se llama cuando se crea una nueva instancia de una clase. En Python, el constructor se define con el método `__init__`.

3. Campos.

Los campos o atributos son variables que pertenecen a una clase o a una instancia de una clase. Pueden ser atributos de clase (compartidos por todas las instancias) o atributos de instancia (específicos para cada objeto).

- **Atributos de instancia:** Definidos dentro del constructor `__init__` y precedidos por `self`.

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre # Atributo de instancia
        self.edad = edad # Atributo de instancia
```

- **Atributos de clase:** Definidos directamente en la clase y compartidos por todas las instancias.

```
class Persona:
    especie = "Humano" # Atributo de clase
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
```

4. Métodos.

Los métodos son funciones definidas dentro de una clase. Pueden operar sobre los atributos de la instancia. En Python se nos presentan 3 tipos de métodos:

- **Métodos de Instancia:** Operan en una instancia de la clase y acceden a los atributos de la instancia usando `self`.

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def saludar(self):
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años.")
```

Para esta instancia

```
persona1 = Persona("Juan", 30)
persona1.saludar()
```

Obtenemos la salida

```
> Hola, mi nombre es Juan y tengo 30 años.
```

- **Métodos de Clase:** Operan en la clase y se definen usando el decorador `@classmethod`. Acceden a los atributos de la clase usando `cls`.

```
class Persona:
    especie = "Humano"
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    @classmethod
    def mostrar_especie(cls):
        print(f"Somos de la especie {cls.especie}")

Persona.mostrar_especie() # Output: Somos de la especie Humano
```

- **Métodos Estáticos:** No operan en una instancia o en la clase y se definen usando el decorador `@staticmethod`.

```
class Persona:
    @staticmethod
    def es_mayor_de_edad(edad):
        return edad >= 18

print(Persona.es_mayor_de_edad(20)) # Output: True
```

II. Manejo de memoria.

En Python, el manejo de memoria es mayormente **implícito**, facilitado por un recolector de basura automático. Sin embargo, también es posible gestionar la memoria explícitamente en ciertas situaciones.

Manejo Implícito de Memoria

Python utiliza un recolector de basura basado en conteo de referencias y recolección de objetos no referenciados.

- **Conteo de referencias:** Cada objeto en Python tiene un contador de referencias que indica cuántas referencias apuntan a él. El contador aumenta cuando una nueva referencia a un objeto se crea y disminuye cuando una referencia a un objeto se elimina. Cuando este contador llega a cero, el objeto se libera automáticamente.
- **Recolección de ciclos:** Algunos objetos pueden formar ciclos (A referencia a B y B referencia a A). El conteo de referencias no puede liberar estos objetos, por lo que Python utiliza un recolector de ciclos para identificar y liberar estos ciclos.
- **Generaciones de objetos:** El recolector de basura agrupa objetos por generaciones, donde los objetos más jóvenes se revisan más frecuentemente que los más antiguos.

Manejo Explícito de Memoria

Python no permite directamente la gestión explícita de la memoria como `new/delete` en C++. Sin embargo, proporciona métodos para controlar ciertos aspectos de la memoria:

- `del`: Este comando se usa para eliminar referencias a objetos, lo que puede ayudar a reducir el uso de memoria.
- Módulo `gc`: Este módulo permite interactuar con el recolector de basura para ajustar la configuración y forzar la recolección.

III. Asociación de métodos.

Python utiliza principalmente la asociación dinámica de métodos. En la asociación dinámica, también conocida como "ligadura tardía" o "late binding," la resolución del método que se va a invocar se realiza en tiempo de ejecución. Esto permite una gran flexibilidad, ya que los métodos pueden ser redefinidos o reemplazados en tiempo de ejecución.

En Python, la resolución de métodos se basa en la estructura del objeto y no en su tipo estático. Esto significa que Python verifica la existencia del método en la instancia y su clase en tiempo de ejecución.

```
class Animal:
    def hacer_sonido(self):
        print("Sonido genérico")

class Perro(Animal):
    def hacer_sonido(self):
        print("Ladrido")

animal = Animal()
perro = Perro()

animal.hacer_sonido()  # Output: Sonido genérico
perro.hacer_sonido()   # Output: Ladrido
```

Aunque Python usa asociación dinámica por defecto, hay formas de forzar un comportamiento similar a la asociación estática, aunque no es habitual y suele ser desaconsejado. Entre estas alteraciones tendriamos:

- **Llamadas explícitas a métodos de clase base**: Usando el nombre de la clase base para llamar a un método específico.

```
class Perro(Animal):
    def hacer_sonido(self):
        # Llama al método de la clase base
        super(Perro, self).hacer_sonido()
        print("Ladrido")

perro = Perro()
perro.hacer_sonido()
# Output:
# Sonido genérico
# Ladrido
```

- **Funciones y métodos estáticos**: Definidos usando los decoradores `@staticmethod` y `@classmethod`, que no dependen de la instancia.

```
class Matematica:
    @staticmethod
    def sumar(a, b):
        return a + b

print(Matematica.sumar(3, 5))  # Output: 8
```

IV. Jerarquía de tipos.

En Python, todas las clases derivan de la clase base `object`, ya sea directa o indirectamente. Esto incluye tanto los tipos incorporados (por ejemplo, `int`, `str`) como las clases definidas por el usuario.

```
# Ejemplo básico de una jerarquía de tipos
class Animal:
    def hacer_sonido(self):
        pass

class Perro(Animal):
    def hacer_sonido(self):
        return "Ladrido"

class Gato(Animal):
    def hacer_sonido(self):
        return "Maullido"
```

Herencia Múltiple

Python soporta la herencia múltiple, lo que significa que una clase puede heredar de múltiples clases base. Esto puede ser útil para combinar comportamientos de diferentes clases, pero también puede complicar la resolución de métodos.

Python utiliza el algoritmo MRO (Method Resolution Order) para resolver el orden de búsqueda de métodos.

```
class A:
    def metodo(self):
        print("Método de A")

class B:
    def metodo(self):
        print("Método de B")

class C(A, B):
    pass

obj = C()
obj.metodo()  # Output: Método de A
```

Polimorfismo Paramétrico

Python implementa polimorfismo paramétrico a través de la biblioteca `typing`, que proporciona soporte para tipos genéricos. Esto permite definir clases y funciones que pueden operar sobre tipos arbitrarios.

Manejo de Varianzas

El manejo de varianzas en Python no es tan explícito como en algunos otros lenguajes como Java o C#. Sin embargo, la librería `typing` proporciona herramientas básicas para trabajar con varianzas en tipos genéricos.