FUNDAMENTOS DE PROGRAMACIÓN
EXAMEN FINAL (30%)
Fecha: 20/05/2024 Hora: 08:00a.m.

Nombre: _____ **Profesor:** Daniel Saldarriaga López **Grupo:** E

LEA CUIDADOSAMENTE TODO EL EXÁMEN ANTES DE COMENZAR A DESARROLLARLO. El profesor no responderá ninguna pregunta durante el examen. Cualquier duda, usted mismo la deberá resolver.

Condiciones: Se pueden usar notas personales, pero no internet. En particular, no se debe usar ningún tipo de servicio de inteligencia artificial.

1. **Estructuras básicas de programación (10%)**

El siguiente código tiene como objetivo calcular el índice de masa corporal de una persona. El IMC se obtiene dividiendo el peso de la persona entre el cuadrado de su altura. Dependiendo del resultado, el código debe retornar una condición de peso de la persona:

- Menos de 18.5: "Desnutrición"
- Entre 18.5 y menos de 25: "Normal"
- Entre 25 y menos de 30: "Sobrepeso"
- 30 o más: "Obesidad"

```
def calcular_imc(peso_kg, altura_m):
    imc = peso_kg / (altura_m ** 2)
    return categoriaIMC(imc)

def categoria_imc(imc):
    if imc > 18.5
        return "Desnutrición"
    elif 18.5 >= imc > 25:
        return "Normal"
    elif 25 >= imc > 30:
        return "Sobrepeso"
    else:
       "Obesidad"

# implementación
calcular_imc(80, 1,8):
```

El código anterior tiene algunos errores. Identifica qué errores tiene el código (de sintaxis, de lógica o de implementación). Analiza el código y explica cómo corregir los errores.

2. **Programación Orientada a Objetos (10%)**

La siguiente clase tiene varios errores de sintaxis. Identifícalos y analiza cómo solucionarlos:

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

    def make_sound(self):
        return NotImplementedError

class Perro(Animal):
    def __init__(self, nombre, edad, raza):
        super().__init__(nombre, edad)
        self.raza = raza

    def sonido(self):
        return "Guau guau"

class Cat(Animal):
    def __init__(self, nombre, edad, raza):
        super().__init__(nombre, edad)
        self.raza = raza

    def hacerSonido(self)
       "Miau miau"
```

3. **Programación Orientada a Objetos (20%)**

Analiza el siguiente código y responde las preguntas a continuación:

```python
class Vehicle:
    def __init__(self, make, model, year):
        self.__make = make
        self.__model = model
        self.__year = year

    def get_make(self):
        return self.__make

    def get_model(self):
        return self.__model

    def get_year(self):
        return self.__year

class Car(Vehicle):
    def __init__(self, make, model, year, num_doors):
        super().__init__(make, model, year)
        self.__num_doors = num_doors

    def get_num_doors(self):
        return self.__num_doors

class Motorcycle(Vehicle):
    def __init__(self, make, model, year, engine_type):
        super().__init__(make, model, year)
        self.__engine_type = engine_type

    def get_engine_type(self):
        return self.__engine_type
```

- ¿Qué conceptos de programación orientada a objetos identificas en el código? Explica cada uno.
- Al código anterior le falta uno de los tres conceptos vistos en clase (herencia, polimorfismo, encapsulamiento). Explica cómo se podría implementar el concepto faltante.

4. **Algoritmos de Ordenamiento (30%)**

A continuación, se presentan dos algoritmos de ordenamiento de arreglos de una dimensión:

```python
def bubble_sort(array):
    n = len(array)
    for i in range(n):
        is_sorted=True
        for j in range(n-1-i):
            if array[j] > array[j+1]:
                array[j], array[j+1] = array[j+1], array[j]
                is_sorted = False
        if is_sorted:
            break

    return array
```

```python
def quick_sort(lista_desordenada):
    low=[]
    same=[]
    high=[]

    if len(lista_desordenada) <2:
        return lista_desordenada

    pivot = random.choice(lista_desordenada)

    for i in lista_desordenada:
        if i<pivot:
            low.append(i)
        elif i>pivot:
            high.append(i)
        else:
            same.append(i)

    return quick_sort(low) + same + quick_sort(high)
```

Analiza y explica:

- Haz un análisis de ambos algoritmos y explica cuál es más eficiente que otro para un gran volumen de datos. Explica las razones que harían que un algoritmo sea más eficiente que otro.
- Selecciona alguno de los dos algoritmos y explica cómo lo modificarías para que también sirva para ordenar arreglos de dos dimensiones.

5. **Algoritmos de Búsqueda (30%)**

El siguiente código en Python tiene como objetivo encontrar un número específico en una lista de números enteros utilizando la búsqueda ternaria. Analiza el código y responde las preguntas a continuación.

```python
def search(numbers, left, right, target):
    if left > right:
        return -1
    third = (right - left) // 3
    mid1 = left + third
    mid2 = right - third
    if numbers[mid1] == target:
        return mid1
    if numbers[mid2] == target:
        return mid2
    if target < numbers[mid1]:
        return search(numbers, left, mid1 - 1, target)
    elif target > numbers[mid2]:
        return search(numbers, mid2 + 1, right, target)
    else:
        return search(numbers, mid1 + 1, mid2 - 1, target)

def ternary_search(numbers, target):
    return search(numbers, 0, len(numbers) - 1, target)

numbers = [1, 3, 4, 5, 6, 8, 10, 12]
target = 5
print(ternary_search(numbers, target))  # Output: 3
```

- ¿Qué hace el código anterior? Describe su funcionamiento.
- Identifica y explica cualquier error o ineficiencia en el código o compáralo con algún otro algoritmo que conozcas y explica si este algoritmo es más eficiente o no, y justifica tu respuesta.