Universidad Nacional de San Martín

Exámen Parcial II

Python

Instrucciones:

- Lee cuidadosamente cada punto antes de programar.
- Cuentan con dos horas para realizarlo.
- Se puede disminuir puntaje de cada punto de acuerdo a cómo esté desarrollado.
- El exámen se aprueba con 50 puntos.

1. Ejercicios de Python

1. Listas y Condicionales

Dada una lista de números enteros, escribe una función llamada `modificar_numeros` que:

- Remueva todos los números impares que estén en posiciones impares de la lista (basado en el índice).
- Devuelva una lista donde cada número par en posición par sea dividido entre 2 si es mayor a 10, o multiplicado por 3 si es menor o igual a 10.

Ejemplo de entrada: `[4, 15, 8, 7, 14, 12, 5]`

Ejemplo de salida: `[6, 4, 7]`

2. Ciclos anidados y Diccionarios

Escribe una función llamada `crear_tablero_valores` que:

- Reciba dos números enteros n y m (número de filas y columnas).
- Devuelva un diccionario donde las claves sean las coordenadas (i, j) y los valores sean números enteros consecutivos, comenzando en 1.
- Además, si la posición (i, j) tiene ambos índices pares, el valor se multiplica por 2.

Ejemplo de entrada: $n = 3$, $m = 2$

Ejemplo de salida: {`(0,0): 2, (0,1): 2, (1,0): 3, (1,1): 4, (2,0): 10, (2,1): 6`}

3. Conjuntos

Crea una función llamada `analizar_sets` que reciba dos listas de números enteros. Convierte ambas listas en conjuntos y realiza lo siguiente:

- Devuelve el conjunto que contiene los elementos comunes entre las dos listas.

- Luego, devuelve el conjunto resultante de la diferencia simétrica entre los dos conjuntos.
- Finalmente, crea un conjunto con los valores únicos de los conjuntos originales que sean múltiplos de 4 y que no estén en la intersección.
- Todos los resultados deben ser devueltos como una tupla de conjuntos.

Ejemplo de entrada: `[1, 2, 3, 4, 8]`, `[3, 4, 5, 8, 12]`

Ejemplo de salida: `({3, 4, 8}, {1, 2, 5, 12}, {12})`

4. Decoradores con múltiples funciones

Escribe un decorador llamado `multiplicar_si_mayor` que reciba un número como parámetro. Este decorador debe aplicarse a dos funciones:

- Una función `suma(a, b)` que sume dos números.
- Una función `resta(a, b)` que reste dos números.

El decorador debe verificar si el resultado de la función es mayor a 10 y, si es así, multiplicarlo por el valor proporcionado al decorador. Implementa el decorador y aplica ambas funciones.

Ejemplo de uso:

```
@multiplicar_si_mayor(4)
def suma(a, b):
    return a + b

@multiplicar_si_mayor(3)
def resta(a, b):
    return a - b
```

Si se llama a `suma(5, 8)`, el resultado debería ser 52, ya que el decorador multiplica el resultado original 13 por 4.

5. Clases y POO

Diseña una clase llamada `Cilindro` que tenga los siguientes atributos y métodos:

- **Atributos:** `radio` y `altura`.
- **Métodos:**
  - `area_superficie()`: Calcula y devuelve el área de la superficie del cilindro.
  - `volumen()`: Calcula y devuelve el volumen del cilindro.
  - `cambiar_dimensiones(nuevo_radio, nueva_altura)`: Cambia el radio y altura del cilindro.

Nota: El área de la superficie de un cilindro es $2\pi r(r+h)$ y el volumen es $\pi r^2 h$.

6. Pilares de la Programación Orientada a Objetos

En base al siguiente código:

```
from abc import ABC, abstractmethod

class Vehiculo(ABC):
    @abstractmethod
    def tipo(self):
        pass

class Auto(Vehiculo):
    def tipo(self):
        return "Automóvil"

class Bicicleta(Vehiculo):
    def tipo(self):
        return "Bicicleta"
```

Explica:

- ¿Cómo se implementa la abstracción en el código? ¿Qué ventaja ofrece este enfoque?
- ¿Qué ventaja tiene el polimorfismo en este caso?
- Explica cómo se podría aplicar la herencia para agregar una clase `Camion` que extienda la funcionalidad de la clase base `Vehiculo`, añadiendo un atributo propio como `carga_maxima`.