UNIVERSIDAD ABIERTA INTERAMERICANA
FACULTAD DE TECNOLOGÍA INFORMÁTICA
MATERIA: PROGRAMACIÓN II
1er Parcial
Teórico y Práctico

Alumno: Lastra, Julián Marcos. **Fecha: 04/10/22** **Tema 1**
Comisión – Localización – Turno: Noche
Práctica: **Teoría:** **Nota:**

Temas para evaluar: Paradigmas: Estructurado y POO, introducción al lenguaje python.
Objetivos:
Comprender las pautas del proceso para la creación de software desde la perspectiva de la Ing. SW.
Comprender cómo se aplican las técnicas del paradigma estructurado y POO en el lenguaje pyhton

Modalidad: Parcial domiciliario
Requisitos para aprobar: Para que el parcial esté aprobado el alumno deberá tener correctamente desarrolladas el 60% de la teoría y resuelto el ejercicio práctico.
Tiempo:
Recomendaciones:
a) Lea todo el parcial antes de comenzar a responder.
b) Desarrolle una redacción clara y precisa contestando lo que la pregunta requiere.
c) Observe la ortografía ya que la misma es parte del parcial.
d) Si considera que no comprende alguna consigna antes de comenzar consulte a su profesor.
Notas: Las preguntas en las que se seleccionen opciones se deberá optar solo por una de las posibilidades. La indicación se efectuará con una X sobre su lateral izquierdo, será considerada nula si presenta tachaduras o enmiendas.
Las preguntas que solicitan justificación serán consideradas válidas si poseen la misma correctamente.
Las preguntas de múltiples posibilidades y verdadero / falso restan 0.50 puntos en caso de estar mal contestadas. En las preguntas verdadero / falso se debe tachar la opción incorrecta.

(*) la cifra entre paréntesis en cada pregunta es la cantidad de puntos sobre 100.

1. Indique tres diferencias sustanciales al momento de diseñar un programa utilizando el paradigma estructurado y paradigma orientado a objetos. (10)

En el paradigma estructurado:
- La cantidad de líneas de código suele ser mayor que en otros paradigmas dado el tipo de escritura.
- Complejidad en la reutilización, es mas difícil diseñar soluciones reuitilzables.
- Se diseñan en mono-bloque de código lo que da mayor complejidad de agregar nuevas funcionalidades sin producir daño colateral.

En el paradigma orientado a objetos:
- Se combinan los datos y los procedimientos en una entidad única.
- Aumento de los requerimientos de memoria y procesos en general con respecto a otros paradigmas, por lo que hay que tener cuidado al diseñar nuestras soluciones.
- Se necesita una documentación clara ya que las visiones de las abstracciones no suelen estar unificadas entre diferentes diseñadores

2. ¿Por qué decimos que python es un lenguaje semi-interpretado y con tipado dinámico? (10)

Por el lado de semi-interpretado lo decimos porque Python tiene muchas de las características de los lenguajes compilados. En Python, como en C#, Java y otros lenguajes, el código fuente primero se traduce a un pseudo código máquina intermedio llamado bytecode la primera vez que se ejecuta, generando archivos .pyc o .pyo.

Con respecto a tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo .

```
#Ejercicio 2
a = 1 #En C# para hacer esto tengo que declarar el tipo: int a;
a = "uno" #Como vemos podemos cambiar el tipo de la variable.
```

3. ¿Qué estructuras condicionales permite el lenguaje python? Ejemplifique con una porción de código. (10)

```
#Python permite las siguientes estructuras condicionales: IF - ELSE - ELIF.
a = 1
if a == 1: #Uso del IF
    print("A = 1")
elif a == 2: #Uso del ELIF - Mas conocido como ELSE IF en otros lenguajes.
    print("A = 2")
else: #Uso del ELSE
    print("La condicion es falsa")
```

4. ¿Qué estructuras repetitivas permite el lenguaje python? Ejemplifique con una porción de código. (10)

```python
#Ejercicio 4
#Python permite las siguientes estructuras repetitivas: WHILE - FOR
from datetime import datetime
from time import sleep
now = datetime.now()
listCantantes = {"Cerati", "Spinetta","Plant","Jagger"}
for cantante in listCantantes:
    print(f"Nombre del cantante: {cantante}")
while True:
    print(f"La hora es: {now.strftime('%H:%M:%S')}")
    sleep(1)
```

5. ¿Cuáles son las diferencias entre listas, tuplas y diccionarios en python? (10)

Listas: es una colección de datos ordenada y que puede ser modificada. Esta permite valores duplicados.
Diccionario: es una colección de datos ordenada y que puede ser modificada. No permite duplicados.
Tuplas: al ser indexadas te permiten tener "ítems" con el mismo valor. Los ítems de la tupla no pueden ser modificados.

En el ejemplo vemos como funcionan los diferentes tipos. Podemos ver que listas y tuplas permiten elementos repetidos. A su vez, vemos como el diccionario, al tener una clave y un valor únicos, por lo tanto no se pueden agregar duplicados.
Otra diferencia clara es al momento de declarar lo que hay que usar para cada una "[ ]", "( )" y "{ }".

```python
#Ejercicio 5.
listAutosDeportivos = ["BMW M3","BMW M3","Ford Mustang"] #Lista
dictAutosDeportivos = {} #Diccionario
dictAutosDeportivos["BMW"] = "M3"
dictAutosDeportivos["Ferrari"] = "Testarossa"
dictAutosDeportivos["Ford"] = "Mustang"
tuplaAutos = ("Nissan", "Nissan", "Ford", "BMW") #Tupla
```

6. ¿Cómo se declara una clase? Ejemplifique (10)

```python
#Ejercio 6
#Para declarar una clase, hay que utilizar la palabra reservada "class"
class Auto:
```

7. ¿Cómo declaro una propiedad? Ejemplifique (10)

```python
class Auto:
    marca = "Nissan" # Propiedad 1
    modelo = "Skyline" # Propiedad 2
    año = 1996 # Propiedad 3
    #Para declarar las propiedades simplemente definimos una "variable".
```

8. ¿Qué es el Duck typing? (10)

Duck typing significa que una operación no especifica formalmente los requisitos que deben cumplir sus operandos, sino que simplemente lo prueba con lo que se le da. Si uno de estos no puede cumplir con los requisitos al momento de la ejecución se va a producir una Exception (except).

9. Ejemplifique cómo se puede implementar polimorfismo con python (10)

```python
#Ejercicio 9.
#Dado que no requerimos declarar el tipo, simplemente el polimorfismo se declara:
def volar(loQueVuela):
    print(f"Esta volando {loQueVuela}")
class Animal:
    nombre = "Pato"
class Avion:
    modelo = "737"
volar(Animal)
volar(Avion)
```

10. ¿Cuál es la diferencia entre los métodos init y new? (10)

__new__ se usa cuando tenes que controlar la creación de una nueva instancia, en cambio __init__ se usa cuando necesitas controlar la inicializacion de una nueva instancia.

__new__ es el primer paso de la creación de la instancia. Se llama primero y es el responsable de retornar una nueva instancia de tu clase.
En cambio, __init__ no retorna nada; solo es responsable por inicializar la instancia después de que fue creada.

UNIVERSIDAD ABIERTA INTERAMERICANA
FACULTAD DE TECNOLOGÍA INFORMÁTICA
MATERIA: PROGRAMACIÓN II
1er Parcial
Teórico y Práctico

PRÁCTICA

1. Escribir una clase **Personaje** que contenga los atributos **vida**, **posicion** y **velocidad**, y los métodos **recibir_ataque**, que reduzca la vida según una cantidad recibida y escriba un mensaje por consola si la vida pasa a ser menor o igual que cero, y **mover** que reciba una dirección y se mueva en esa dirección la cantidad indicada por velocidad. (20)
2. Escribir una clase **Soldado** que herede de **Personaje**, y agregue el atributo **ataque** y el método **atacar**, que reciba otro personaje, al que le debe hacer el daño indicado por el atributo **ataque**. (20)
3. Escribir una clase **Campesino** que herede de **Personaje**, y agregue el atributo **cosecha** y el método **cosechar**, que devuelva la cantidad cosechada. (20)
4. Los soldados pueden atacar a soldados o campesinos. No se debe permitir que un campesino ataque a otro campesino o a un soldado. (10)
5. Genere código python para probar la solución propuesta con la cantidad de personajes que crea conveniente y la prueba del mensaje cuando los personajes tengan vida 0 o se les indique moverse, atacar o cosechar. (30)

Adjunto se encuentra el código correspondiente.

```
#Punto 1
class Personaje:
  vida = 10
  posicion = 0
  velocidad = 24
  def recibir_ataque(self,damage):
    self.vida -= damage
    if self.vida <= 0:
      print("La quedo")
    else:
      print(f"Le quedan {self.vida} puntos de vida al personaje")
  def mover(self,direccion):
    if direccion == "derecha":
      self.posicion += self.velocidad #Muevo el eje X positivamente
      print(f"El personaje se movio a la derecha y ahora se encuentra en la posicion
{self.posicion}")
    if direccion == "izquierda":
      self.posicion -= self.velocidad #Muevo el eje Y negativamente
      print(f"El personaje se movio a la izquierda y ahora se encuentra en la posicion
{self.posicion}") #{type(self).__name__}
#Punto 2
class Soldado(Personaje):
  ataque = 4
  def atacar(self, victima):
    victima.recibir_ataque(victima,self.ataque)
    #print(f"El personaje ahora tiene {victima.vida} puntos de vida restante") #Spamea
banda
#Punto 3
class Campesino(Personaje):
  cosecha = "Maiz"
  cantidadCosechada = 10
  def cosechar(self):
    self.cantidadCosechada += 10
    print(f"El campesino tiene cosechado {self.cantidadCosechada} de {self.cosecha}")
```

```
#Punto 5 y Punto 4.
print("#########################################################")
print("Programacion II - Lastra Julian - Parcial 1")
print("#########################################################")
print("AGE OF EMPIRES 0.1")
print("#########################################################")

soldier = Soldado
farmer = Campesino

#Cosechar
print("El campesino esta cosechando.")
farmer.cosechar(farmer)
print("El campesino esta cosechando.")
farmer.cosechar(farmer)

#Mover
print("Muevo al soldado.")
soldier.mover(soldier,"derecha") #Se mueve a la derecha
print("Muevo al soldado.")
soldier.mover(soldier,"izquierda") #Se mueve a la izquierda
print("Muevo al soldado.")
soldier.mover(soldier,"izquierda") #Se mueve a la izquierda
print("Muevo al soldado.")
soldier.mover(soldier,"izquierda") #Se mueve a la izquierda

#Atacar
print("El soldado ataco a un granjero")
soldier.atacar(soldier,farmer) #El soldado ataca al campesino una vez
print("El soldado ataco a un granjero")
soldier.atacar(soldier,farmer) #El soldado ataca al campesino por segunda vez
print("El soldado ataco a un granjero")
soldier.atacar(soldier,farmer) #El soldado mata al campesino

#Punto 4
#El campesino al no tener el metodo atacar, no puede realizar esta accion
try:
  farmer.atacar(farmer,soldier)
except Exception as ex:
  print("El personaje selccionado para realizar el ataque no puede hacerlo.")
#end Punto 4

print("####################     END     ########################")
```

Firma del Alumno