Instituto Superior de Formación Técnica Nº 151

Carrera: Analista en Sistemas

X Año. Algoritmos y Estructuras de Datos X.

Trabajo Práctico Nº2	Unidad X3- Python POO
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo individual.
Metodología de Desarrollo: Det. docente	**Metodología de Corrección:** Via Classroom.
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

INTRODUCCION A PYTHON.

1. Introducción a POO en Python

Marco Teórico:

1.1 ¿Cómo se define una clase en Python?.
1.2 Que es un atributo de Clase y uno de Instancia, cuales son las diferencias.
1.3 ¿Como se define un constructor en Python?
1.4 ¿Que es un Decorados en Python?
1.5 Que es un Método de Instancia, clase y Estático.
1.6 Dar ejemplos de atributos y métodos en Python

Marco Practico: Realizar en Python

1.7 Crear una Clase en Python
1.8 Asignarles atributos de clase e instancia
1.9 Asignar métodos de Instancia, Clase y estáticos
1.10 Realizar una calculadora que consuma estos métodos

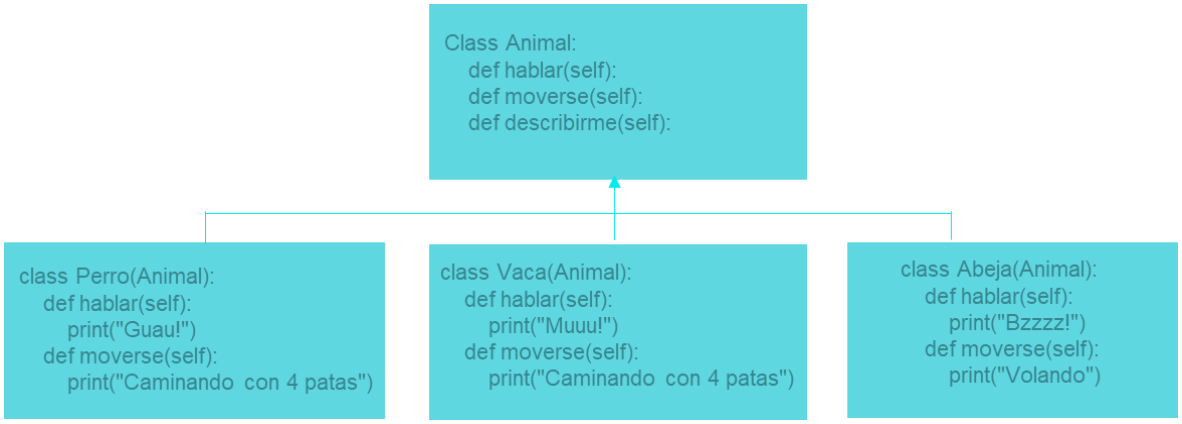
2. Herencia en Python.

Marco teórico

2.1 Como Implementa la Herencia Python, dar ejemplos.
2.2 ¿Para que sirven los métodos “Mágicos” __bases__ y __subclases__ ?
2.3 Como se Implementa la Extensión y modificación de métodos.
2.4 Para que sirve la palabra super()
2.5 ¿Como implementa Python la Herencia múltiple?
2.6 Que es el MRO o Method Resolution Order, para que sirve?

Marco Practico: Realizar en Python

2.1 Desarrollar una APP por consola que Cree una estructura como la de la figura:



2.2 Implementar la Herencia
2.3 Implementar __bases__ y __subclases__
2.4 Implementar base()
2.5 Crear Objetos y mostrar por consola.

3. Decoradores en Python.

Marco Teórico

3.1 Que función tiene @property, que emula?
3.2 Que relación tiene @property con encapsulamiento?
3.3 Que relación tiene @property con Set() y Get()?
3.4 ¿Para que sirve y se utiliza "__" antes de una variable?
3.5 Que relación tiene con el método setter()

Marco Practico
Para la App Anterior

3.1 Implementar dos métodos en alguna Clase y usar @property
3.2 Implementar Encapsulamiento vía "__"
3.3 Implementar los setter()

4. Conceptos POO Implementados en Python.

Marco teórico

4.1 Explicar la cita: La abstracción es un término que hace referencia a la ocultación de la complejidad intrínseca de una aplicación al exterior, centrándose sólo en cómo puede ser usada, lo que se conoce como interfaz.

4.2 Explicar: El acoplamiento en programación (denominado coupling en Inglés) es un concepto que mide la dependencia entre dos módulos distintos de software, como pueden ser por ejemplo las clases.
4.3 ¿A que s ele denomina un “efecto mariposa” en programación?
4.4 ¿Qué entiende por Cohesión?, dar ejemplos.
4.5 ¿Qué relación hay entre Acoplamiento y Cohesión?
4.6 ¿Que es el Encapsulamiento?
4.7 ¿Como implementa Python el encapsulamiento?

Marco Practico

4.1 Revisar la App anterior y tratar de Implementar los conceptos anteriores.
comentar los Cambios en el Código.

5. Polimorfismo en Python

Marco teórico

5.1 ¿Que entiende por polimorfismo?
5.2 Como se implementa el Polimorfismo en Lenguajes estáticos
5.3 Como se implementa el Polimorfismo en Lenguajes dinámicos

Marco Practico

5.1 analizar y describir le siguiente código:

```
for animal in Perro(), Gato():
   animal.hablar()
# Guau!
# Miau!
```

6. Clases Abstractas e Interfaces en Python

6.1 Que entiende por Interface (conceptualmente
6.2 Que diferencia y que relación tiene con la Implementación.
6.3 Que tipos de Interfaces tiene Python.
6.4 que es una Interfaz “Informal”, como se Implementa.
6.5 Que problemas puede ocasionar la implementación de Interfaces Informales?
6.6 Que son las Interfaces Formales?
6.7 Como se implementan las ABC?
6.8 Para que sirve el decorador @abstractmethod?
6.9 Que entiende por clase virtual en Python?
6.10 como se define una clase virtual dar ejemplo.

6.11 Explicar la Librería ABC para colecciones, que permite crear interfaces de colecciones, explicar le ejemplo de la Teoria.

Marco Practico:

6.1 Desarrollar una App que permita a un cliente, manejar distintos drones: Tricópteros. Cuadricópteros. Hexacópteros. Octocópteros. Coaxiales.
6.2 Implementar interfaz Abstracta que permita al Cliente pilotear (despegar, aterrizar, acelerar, frenar, doblar derecha-izquierda, sacar fotos)

7. DuckTyping en Python

Marco Teorico

7.1 Explicar el concepto que subyace a la frase: “If it walks like a duck and it quacks like a duck, then it must be a duck”
7.2 Que no dice el autor: Don’t check whether it is-a duck: check whether it quacks-like-a duck, walks-like-a duck, etc, etc, depending on exactly what subset of duck-like behavior you need to play your language-games with. (comp.lang.python, Jul. 26, 2000) — Alex Martell
7.3 Que significa que a , a Python le dan igual los tipos de los objetos, lo único que le importan son los métodos.
7.4 Explicar el siguiente Código:

```
lista = [Perro(), Gato(), Vaca()]
for animal in lista:
    animal.hablar()
```

Marco Practico

7.1 crear una App que genere animales con métodos iguales (implementados de diferente manera), cargarlos a una lista y ejecutar un método usando las bondades del Duck Typing.