

```
study = UC(name = "PDA", studyHs = 8)
mate = Mate() #nuevo mate
```

```
while study.notDone():
    study.meterle()
    mate.cebar()
    if mate.termoEmpt():
        mate.refill()
```

Programación Digital Avanzada

Módulo 3 - POO

Mag. Bioing. Baldezzari Lucas

Ingeniería Biomédica
7mo Semestre

2022

```

class ArduinoCommunication:
    """Clase para comunicación entre Arduino y PC utilizando la libreria PySerial
    Constructor del objeto ArduinoCommunication

    Parametros
    -----
    port: String
        Puerto serie por el cual nos conectaremos
    trialDuration: int
        Duración total de un trial [en segundos]
    stimONTime: int
        Duración total en que los estímulos están encendidos
    timerFrequency: int
        Variable para "simular" la frecuencia [en Hz] de interrupción del timer

    """
    def __init__(self, port, trialDuration = 6, stimONTime = 4,
                  timerFrequency = 1000, timing = 1, useExternalTimer = False,
                  ntrials = 1):

        self.dev = serial.Serial(port, baudrate=19200)

        self.trialDuration = int((trialDuration*timerFrequency)/timing) #segundos
        self.stimONTime = int((stimONTime*timerFrequency)/timing) #segundos
        self.stimOFFTime = int((trialDuration - stimONTime)/timing*timerFrequency)
        self.stimStatus = "on"
        self.trial = 1
        self.trialsNumber = ntrials

        self.movements = [b'0',b'1',b'2',b'3',b'4',b'5'] #lista con los comandos
        """
        movements:
        b'0' = STOP (Neurorace) / ADELANTE (Mentalink)
        b'1' = ADELANTE (Neurorace) / 45° ADELANTE E IZQUIERDA (Mentalink)
        b'2' = LEFT (Neurorace) / IZQUIERDA (Mentalink)
        b'3' = ATRAS (Neurorace) / ATRAS (Mentalink)
        b'4' = DERECHA (Neurorace) / DERECHA (Mentalink)
        b'5' = 45° ADELANTE Y DERECHA (Mentalink)
        #El STOP de mentalink será self.moveOrder = b'63' (0b00111111)
        """

        self.sessionStatus = b"1" #sesión en marcha
        self.stimuliStatus = b"0" #los estímulos empiezan apagados
        self.moveOrder = self.movements[0] #EL robot empieza en STOP
        self.estadoRobot = 0
        # self.moveOrder = b'63' #El STOP de mentalink será self.moveOrder =
        """

```

Veremos una Introducción a la POO

- ¿Qué es?
- Clases y Objetos
- Atributos y Métodos
- Encapsulamiento
- Herencia
- Polimorfismo
- Fundamentos de UML

POO - ¿Qué es?

Programación **O**rientada a **O**bjetos (**O**riented **O**bject **P**rogramming, **OOP**).

Paradigma de programación creado en 1970.

La idea detrás de la POO es escribir **clases que representan cosas y/o situaciones del mundo real**.

Estas clases poseen **atributos** y **métodos**.

A partir de una clase creamos **objetos**.

Los objetos poseen la capacidad de **interactuar** con objetos del mismo tipo y con otros objetos.

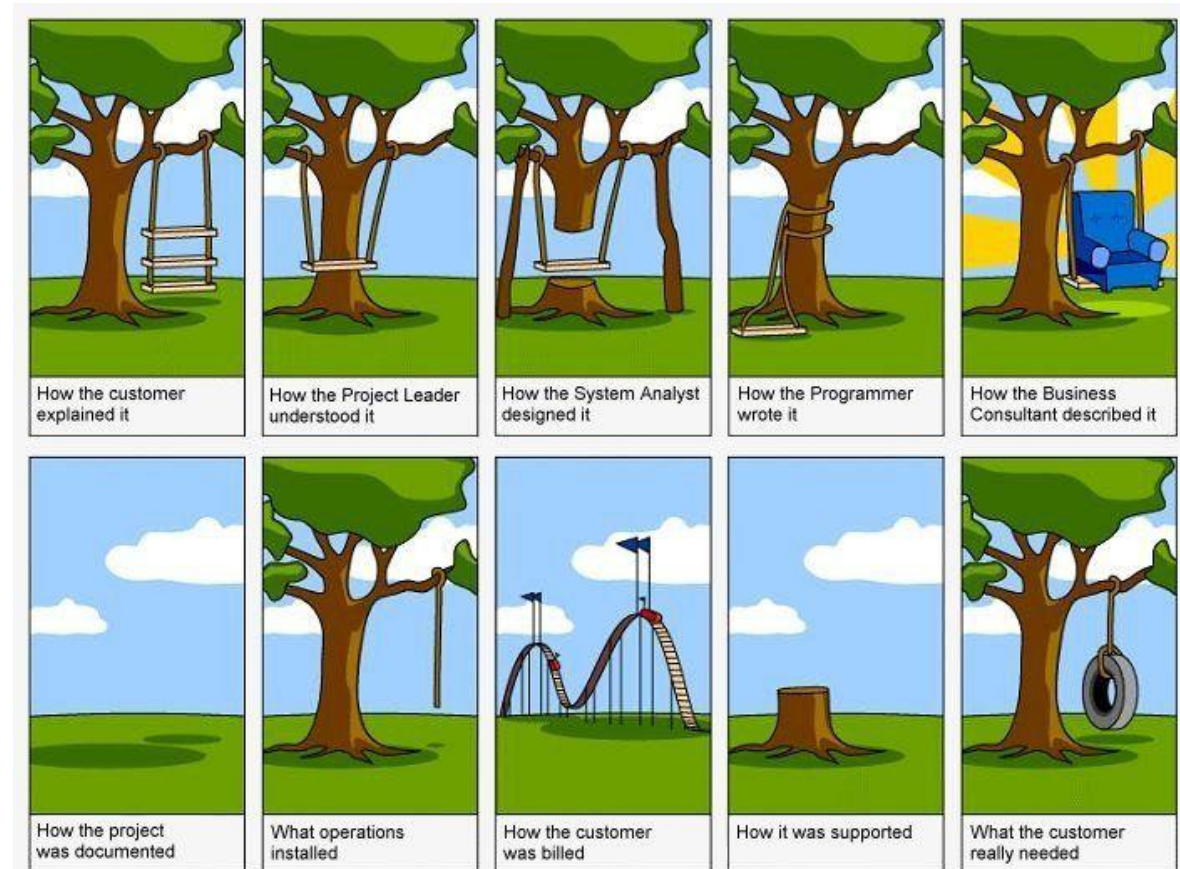
La POO nos permite modelar cosas del mundo real y como estas interactúan entre sí, tales como estudiantes y profesores, vehículos, animales, emails, etc.

POO – Tercera etapa del diseño de software

La POO es la tercera etapa en el proceso del desarrollo de software.

De manera ideal, se tiene:

1. Análisis Orientado a Objetos
2. Diseño Orientado a Objetos
3. Programación Orientada a Objetos



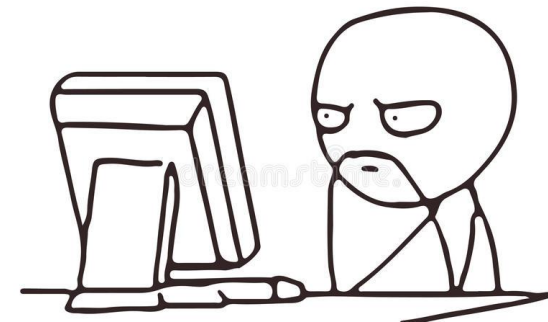
OOA

El **OOA** es el proceso de **identificar objetos** y la **interacción** entre estos a partir del planteo de un **problema**, de la **observación** de un sistema o **tarea** que alguien quiere convertir en una aplicación.

El OOA busca saber **qué** se necesita hacer. De esta etapa se obtienen los **requerimientos**.

Ejemplo: Podríamos querer realizar un “Software para diseño, cálculo y monitoreo de redes de gases medicinales”, a partir de este planteo definimos los “requerimientos”.

Como requerimientos podríamos mencionar...

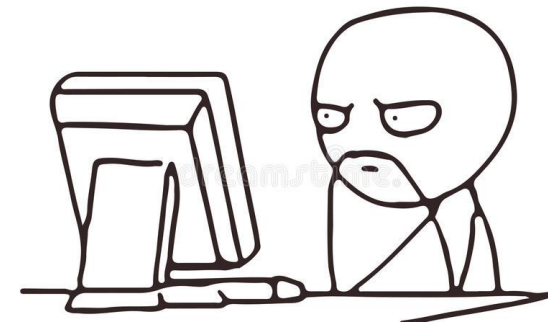


Como requerimientos podríamos mencionar,

- Cálculo de diámetros de tuberías de oxígeno, aire comprimido y vacío para diferentes áreas en base a cantidades de cama, áreas, etc.
- Monitoreo de consumo en las diferentes partes del circuito de la red de gases.
- Etc...

De lo cual podríamos tener los siguientes objetos,

- Cañerías de oxígeno, AC y vacío.
- Áreas y/o servicios.
- ¿Qué más...?



OOD

Proceso de convertir los requerimientos de la etapa OOA en especificaciones de implementación.

Se definen nombres de objetos, atributos, comportamientos (métodos) y cómo van a interactuar entre ellos.

Es una etapa acerca de *cómo* se harán las cosas.

Se convierten los requerimientos del OOA en clases e interfaces que (idealmente) pueden ser implementadas en un lenguaje de OOP.

Ejemplo:

Nombre de clases: Clases “Caño de O2”, “Caño de Vacío” y “Caño de AC”.

Atributos y métodos: ¿?

```

class ArduinoCommunication:
    """Clase para comunicación entre Arduino y PC utilizando la libreria PySerial
    Constructor del objeto ArduinoCommunication

    Parametros
    -----
    port: String
        Puerto serie por el cual nos conectaremos
    trialDuration: int
        Duración total de un trial [en segundos]
    stimONTime: int
        Duración total en que los estímulos están encendidos
    timerFrequency: int
        Variable para "simular" la frecuencia [en Hz] de interrupción del timer

    """
    def __init__(self, port, trialDuration = 6, stimONTime = 4,
                 timerFrequency = 1000, timing = 1, useExternalTimer = False,
                 ntrials = 1):

        self.dev = serial.Serial(port, baudrate=19200)

        self.trialDuration = int((trialDuration*timerFrequency)/timing) #segundos
        self.stimONTime = int((stimONTime*timerFrequency)/timing) #segundos
        self.stimOFFTime = int((trialDuration - stimONTime)/timing*timerFrequency)
        self.stimStatus = "on"
        self.trial = 1
        self.trialsNumber = ntrials

        self.movements = [b'0',b'1',b'2',b'3',b'4',b'5'] #lista con los comandos
        """
        movements:
            b'0' = STOP (Neurorace) / ADELANTE (Mentalink)
            b'1' = ADELANTE (Neurorace) / 45° ADELANTE E IZQUIERDA (Mentalink)
            b'2' = LEFT (Neurorace) / IZQUIERDA (Mentalink)
            b'3' = ATRAS (Neurorace) / ATRAS (Mentalink)
            b'4' = DERECHA (Neurorace) / DERECHA (Mentalink)
            b'5' = 45° ADELANTE Y DERECHA (Mentalink)
            #El STOP de mentalink será self.moveOrder = b'63' (0b00111111)
        """

        self.sessionStatus = b"1" #sesión en marcha
        self.stimuliStatus = b"0" #los estímulos empiezan apagados
        self.moveOrder = self.movements[0] #EL robot empieza en STOP
        self.estadoRobot = 0
        # self.moveOrder = b'63' #El STOP de mentalink será self.moveOrder =
        """

```

- Clases
- Objetos
- Atributos

Clases y Objetos

¿Qué es una **Clase**?

En el ámbito de la OOP,

“...una Clase es una plantilla o estructura que contiene atributos y métodos que nos permiten crear objetos de una misma clase...”

Al crear una clase se crea un *nuevo tipo de objeto*. Una clase define una estructura de datos.

Como ejemplo podríamos pensar en la clase *Perro*, la cual especifica que se necesita una raza, una serie de colores, un nombre y una edad para cada perro, pero no contiene ninguno de estos datos a priori.

Clases y Objetos

¿Qué es un **Objeto**?

En el ámbito de la OOP,

“...un Objeto es una entidad que posee un estado (atributos) y un comportamiento (métodos)...”

Un objeto es una instancia de una clase.

Podríamos pensar en tener diferentes perros (objetos). Cada perro tiene sus propias características y sus propios comportamientos, pero todos pertenecen a la clase Perro().

Cuando generamos una instancia de la clase Perro(), estamos creando un objeto Perro() que contiene un nombre, una edad, una raza y colores que los diferencian entre sí.

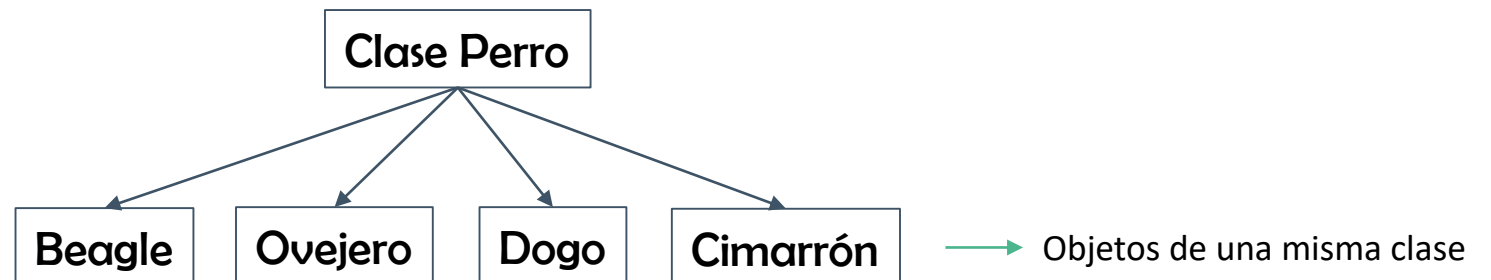
Clases y Objetos

¿Qué es un **Objeto**?

En el ámbito de la OOP,

“...un Objeto es una entidad que posee un estado (atributos) y un comportamiento (métodos)...”

Cuando generamos una instancia de la clase Perro(), estamos creando un objeto Perro() que contiene un nombre, una edad y colores que los diferencian entre sí.



Atributos

Los **atributos** son **características** propias **de cada objeto** y que los diferencian entre sí (nombre, altura, colores, etc).

Cuando creamos una clase especificamos qué atributos posee.

Cuando instanciamos un objeto, debemos especificar todos o algunos de los atributos de la clase.

Recordemos qué...

Instancia de una Clase = Objeto

Atributos

Los atributos se almacenan en **variables**, por lo tanto pueden contener datos primitivos (int, float, bool, string), estructuras de datos complejas (listas, diccionarios, tuplas) o incluso otros Objetos.

Existen:

- **Atributos de instancia**, las cuales toman valores cuando se crea el objeto, *instanciación*.
- **Atributos de clase**, son variables que se aplican a todas las instancias (objetos) creados. Por ejemplo, el número de patas en un perro.

Usualmente se dice que los **atributos** son **seteables**, mientras que las **propiedades** son de **solo lectura** (NOTA: recordar que toda variable en Python es *pública*).

Métodos = Comportamientos

El **comportamiento** de una clase se implementa mediante **métodos** que no son otra cosa que **funciones** propias de la clase, también llamadas **funciones miembro**.

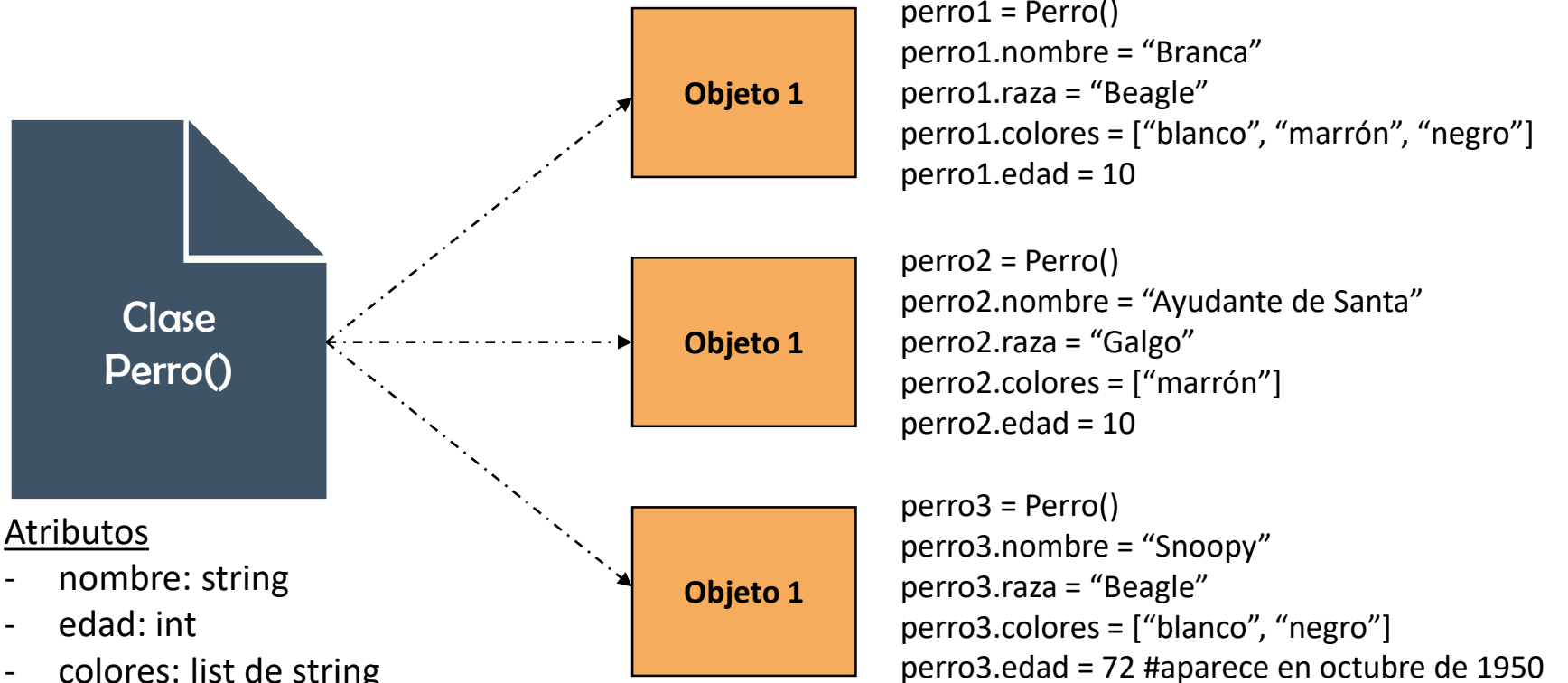
Los métodos llevan a cabo tareas determinadas que permiten que los objetos **interactúen** entre sí o con otros objetos.

Al igual que antes tenemos,

- **Métodos de instancia**, son accesibles cuando se crea el objeto. Estos métodos tienen acceso a cualquier atributo y método, tanto de clase como de instancia.
- **Métodos de clase**, pueden ser utilizados sin crear una instancia de la clase. Tienen accesos a atributos y métodos de clase.
- **Métodos estáticos**, también accesibles sin necesidad de instanciar (crear) un objeto. Estos métodos **no** tienen acceso a atributos ni otros métodos de la clase.

Clases y Objetos

Un **objeto** es una **instancia** de una **clase**.



Atributos

- nombre: string
- edad: int
- colores: list de string
- raza: string

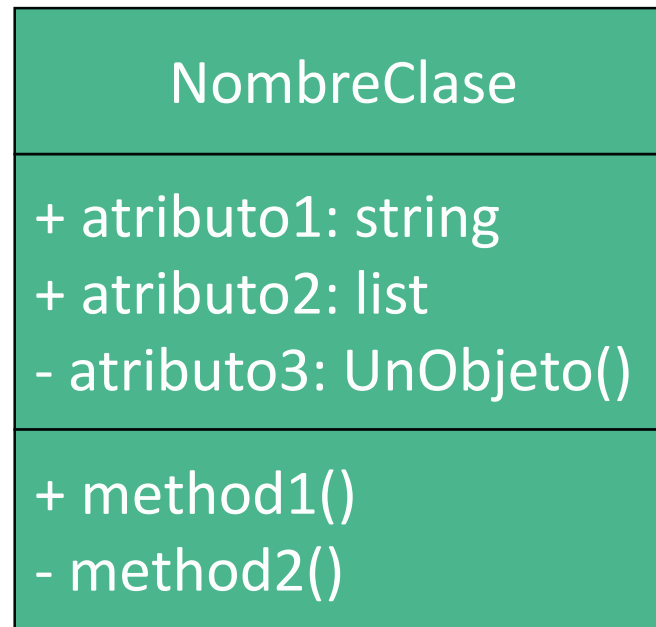
Métodos

- ladrar()
- caminar()
- gruñir()
- comer()

NOTA: Cada objeto posee sus propias características pero todos pertenecen a la clase Perro().

Unified Modeling Language (UML)

El UML nos permite generar diagramas sencillos de nuestras clases y cómo se relacionan entre sí dentro de nuestro programa.



El símbolo + indica atributos públicos
El símbolo – indica atributos privados

El símbolo + indica métodos públicos
El símbolo – indica métodos privados

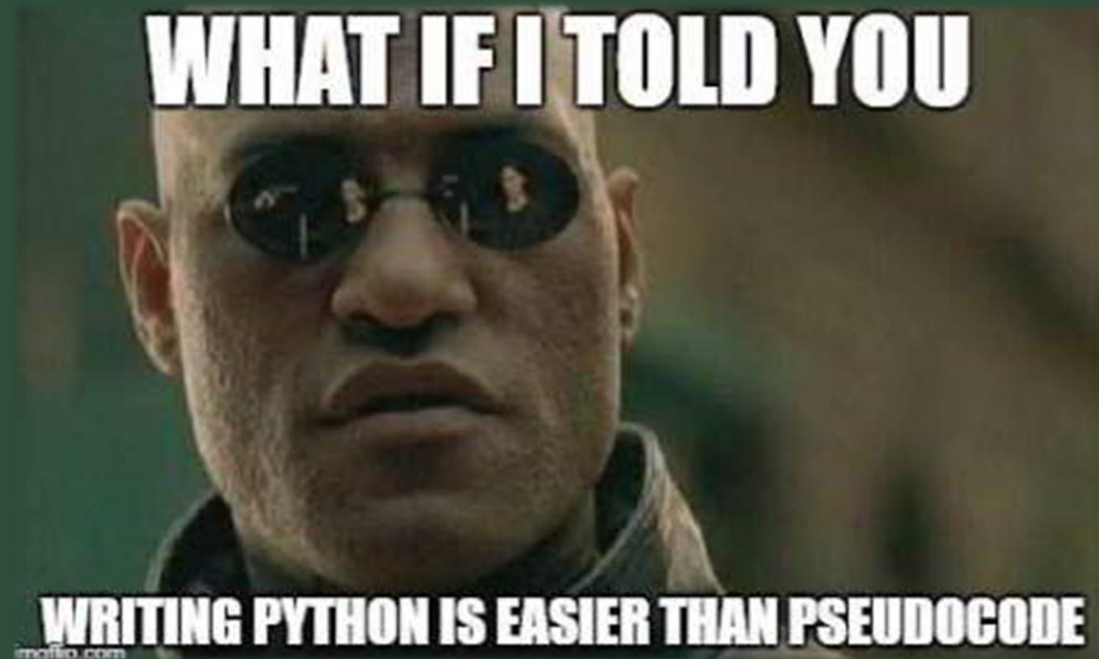
Break 5'



Ejercitación

Realizar el **Ejercicio 1** de la JN titulada “Ejercicios Mod 3 – Teoría”.

20’



```

class ArduinoCommunication:
    """Clase para comunicación entre Arduino y PC utilizando la libreria PySerial
    Constructor del objeto ArduinoCommunication

    Parametros
    -----
    port: String
        Puerto serie por el cual nos conectaremos
    trialDuration: int
        Duración total de un trial [en segundos]
    stimONTime: int
        Duración total en que los estímulos están encendidos
    timerFrequency: int
        Variable para "simular" la frecuencia [en Hz] de interrupción del timer

    """
    def __init__(self, port, trialDuration = 6, stimONTime = 4,
                  timerFrequency = 1000, timing = 1, useExternalTimer = False,
                  ntrials = 1):

        self.dev = serial.Serial(port, baudrate=19200)

        self.trialDuration = int((trialDuration*timerFrequency)/timing) #segundos
        self.stimONTime = int((stimONTime*timerFrequency)/timing) #segundos
        self.stimOFFTime = int((trialDuration - stimONTime)/timing*timerFrequency)
        self.stimStatus = "on"
        self.trial = 1
        self.trialsNumber = ntrials

        self.movements = [b'0',b'1',b'2',b'3',b'4',b'5'] #lista con los comandos
        """
        movements:
            b'0' = STOP (Neurorace) / ADELANTE (Mentalink)
            b'1' = ADELANTE (Neurorace) / 45° ADELANTE E IZQUIERDA (Mentalink)
            b'2' = LEFT (Neurorace) / IZQUIERDA (Mentalink)
            b'3' = ATRAS (Neurorace) / ATRAS (Mentalink)
            b'4' = DERECHA (Neurorace) / DERECHA (Mentalink)
            b'5' = 45° ADELANTE Y DERECHA (Mentalink)
            #El STOP de mentalink será self.moveOrder = b'63' (0b00111111)
        """

        self.sessionStatus = b"1" #sesión en marcha
        self.stimuliStatus = b"0" #los estímulos empiezan apagados
        self.moveOrder = self.movements[0] #EL robot empieza en STOP
        self.estadoRobot = 0
        # self.moveOrder = b'63' #El STOP de mentalink será self.moveOrder =
        """

```

- Abstracción
- Encapsulamiento

Abstracción

¿Cuántos atributos y métodos debe poseer un objeto? ¿Y cuanto de esto debo mostrar al usuario?

La cantidad de atributos y métodos dependerá de qué tan complejo queramos que sea nuestro objeto.

La **abstracción** trata del nivel de detalle **—atributos y métodos—** que queremos modelar del mundo real. Trata de **resaltar lo importante**.

Debemos concentrarnos en **¿Qué hace?** nuestra clase y **no** en **¿cómo lo hace?.**

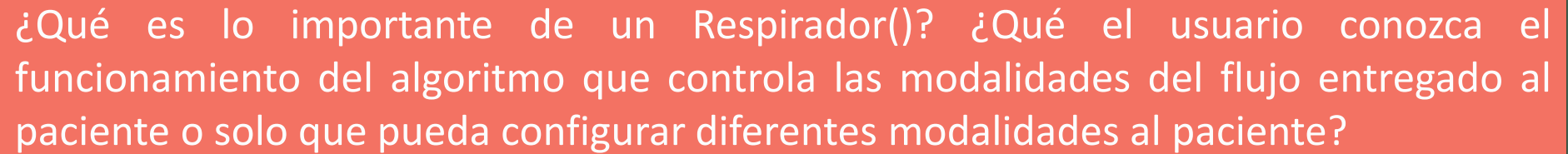
...Menos es más...



Encapsulamiento

El **encapsulamiento** nos permite prescindir de informar de atributos, métodos complejos y la implementación de un objeto para **resaltar lo necesario**.

Encapsular sirve para **ocultar detalles de implementación**.



¿Qué es lo importante de un Respirador()? ¿Qué el usuario conozca el funcionamiento del algoritmo que controla las modalidades del flujo entregado al paciente o solo que pueda configurar diferentes modalidades al paciente?

El encapsulamiento también es una forma de **proteger los datos** de nuestra clase.

Los atributos deberían ser modificados (idealmente) a través de métodos de clase.

Abstracción vs Encapsulamiento

Entonces, ¿cuál es la diferencia entre Abstracción y Encapsulamiento?

La **abstracción** nos define la **complejidad** de nuestra Clase para representar algo del mundo real.

El **encapsulamiento** sirve para **ocultar detalles innecesarios**.



La Abstracción oculta detalles a nivel de diseño,
El Encapsulamiento oculta detalles a nivel de implementación.

Atributos públicos y privados

Python no posee atributos privados.

Existe una filosofía entre las/os desarrolladoras/os de software en Python de hacer un uso responsable de los atributos.

Una variable privada se declara con un guion bajo.

Es posible ocultar o dificultar el acceso de un atributo de clase declarando al mismo con dos guiones bajos delante del nombre de la variable.

```
class MiClase():
    """Esto es el docstring de MiClase"""
    def __init__(self):
        """Constructor de la clase"""
        # Atributos de instancia
        self.atributo1 = 1 # "Público"
        self._atributo2 = 2 # "Lo debemos manejar como atributo privado"
        self.__atributo3 = 3 # "¿Es realmente un atributo privado?"
```

Métodos públicos y privados

Al igual que antes, Python no posee métodos privados.

Al igual que antes, podemos usar un guion o dos guiones antes del nombre de para definir nuestros métodos.

```
class MiClase():
    """Esto es el docstring de MiClase"""
    def __init__(self):
        """Constructor de la clase"""
        # Atributos de instancia
        self.atributo1 = 1 # "Público"
        self._atributo2 = 2 # "Lo debemos manejar como atributo privado"
        self.__atributo3 = 3 # "¿Es realmente un atributo privado?"

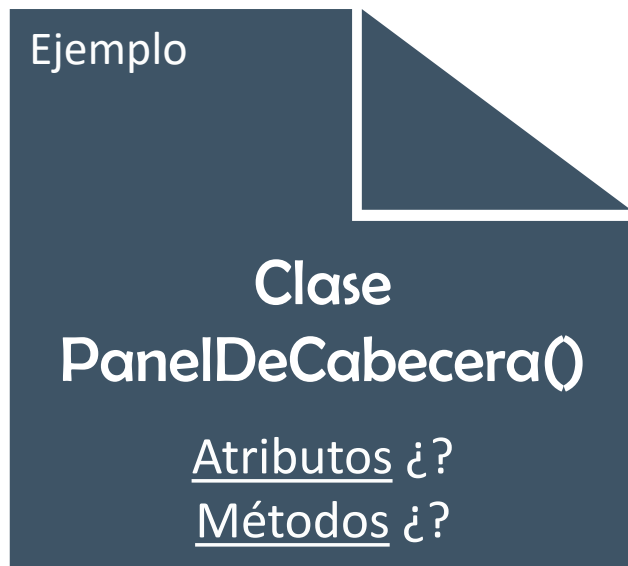
    def _semiprivateMethod(self):
        pass

    def __privateMethod(self):
        pass
```


Ejercicio 1

Supongamos que quisiéramos diseñar un sistema de monitoreo de red de gases medicinales de una Unidad de Cuidados Intensivos. El sistema, como mínimo, debe ser capaz de monitorear e informar la presión y flujo en la cañería de cada gas de cada panel de cabecera que se está utilizando dentro de la UCI.

Proponga clases con atributos y métodos. También defina posibles relaciones entre las diferentes clases propuestas para el sistema de monitoreo. Solo utilice diagramas UML.



¿Definiría otros objetos?
¿Cuáles?

```

class ArduinoCommunication:
    """Clase para comunicación entre Arduino y PC utilizando la libreria PySerial
    Constructor del objeto ArduinoCommunication

    Parametros
    -----
    port: String
        Puerto serie por el cual nos conectaremos
    trialDuration: int
        Duración total de un trial [en segundos]
    stimONTime: int
        Duración total en que los estímulos están encendidos
    timerFrequency: int
        Variable para "simular" la frecuencia [en Hz] de interrupción del timer

    """
    def __init__(self, port, trialDuration = 6, stimONTime = 4,
                  timerFrequency = 1000, timing = 1, useExternalTimer = False,
                  ntrials = 1):

        self.dev = serial.Serial(port, baudrate=19200)

        self.trialDuration = int((trialDuration*timerFrequency)/timing) #segundos
        self.stimONTime = int((stimONTime*timerFrequency)/timing) #segundos
        self.stimOFFTime = int((trialDuration - stimONTime)/timing*timerFrequency)
        self.stimStatus = "on"
        self.trial = 1
        self.trialsNumber = ntrials

        self.movements = [b'0',b'1',b'2',b'3',b'4',b'5'] #lista con los comandos
        """
        movements:
            b'0' = STOP (Neurorace) / ADELANTE (Mentalink)
            b'1' = ADELANTE (Neurorace) / 45° ADELANTE E IZQUIERDA (Mentalink)
            b'2' = LEFT (Neurorace) / IZQUIERDA (Mentalink)
            b'3' = ATRAS (Neurorace) / ATRAS (Mentalink)
            b'4' = DERECHA (Neurorace) / DERECHA (Mentalink)
            b'5' = 45° ADELANTE Y DERECHA (Mentalink)
            #El STOP de mentalink será self.moveOrder = b'63' (0b00111111)
        """

        self.sessionStatus = b"1" #sesión en marcha
        self.stimuliStatus = b"0" #los estímulos empiezan apagados
        self.moveOrder = self.movements[0] #EL robot empieza en STOP
        self.estadoRobot = 0
        # self.moveOrder = b'63' #El STOP de mentalink será self.moveOrder =
        """

```

That's all, folks...

...For today...

```
study = UC(name = "PDA", studyHs = 8)
mate = Mate() #nuevo mate
```

```
while study.notDone():
    study.meterle()
    mate.cebar()
    if mate.termoEmpyt():
        mate.refill()
```

Programación Digital Avanzada

Módulo 3 - POO

Mag. Bioing. Baldezzari Lucas

Ingeniería Biomédica
7mo Semestre

2022