Semana 1

Terminal

- trivial

Git

- trivial

Objetos

Clases y OOP

- nada

```
class Departamento:  # CamelCase notation (PEP8)
    '''Clase que representa un departamento en venta
       valor esta en UF.
    '''
## se puede poner descripcion de la clase asi!
```

Encapsulamiento

- Los objetos pueden tener atributos privados (en términos de Java) y públicos.
- En la vida real, una clase Carpeta puede tener un atributo llamado bloque_de_disco que contiene el bloque donde esta almacenada.
- Esta información no se debería modificar ni leer directamente por una clase distinta, sino que solo debe ser almacenada para uso interno, como por ejemplo al usar funciones como `agregar_archivo(self)`.
- Este encapsulamiento permite abstraer código y ocultar atributos de quienes no los necesitan.

Encapsulamiento e interfaces

- Un objeto clase Auto, puede ofrecer a un objeto clase Conductor metodos como acelerar(), pero no acceso directo al atributo velocidad.
- Si el mismo Auto interactua con un Mecanico, puede ofrecer atributos como nivel_de_aceite, y metodos como cambiar_neumatico(), o abrir_capot()

Encapsulamiento en Python

- En Python no existen los atributos y metodos privados o publicos, pero por convencion se puede *recomendar* que un metodo o atributo sea interno.
- Se hace usando _metodo o _atributo
- Ej rapido:

```
class Mono:
  ''' Clase que modela un mono.
  '''
def __init__(self, hambre, sed, nombre):
  self._hambre = hambre
  self._sed = sed
  self.nombre = nombre

def alimentar(num):
  self._hambre += num

def nivel_hambre():
  return self._hambre

## fuera de la class
mono1 = Mono(10, 11, "mono")
mono1.nombre # bien.
mono1.nivel_hambre() # 10
mono1.alimentar(1)#
```

- En el ejemplo nombre es publico, y se puede usar desde fuera.
- sed y hambre son privados, solo se pueden usar mediante metodos definidos por la clase.

Variables de clase

- Se pueden crear variables que sean globales a todas las instancias de una clase.
- ej(pendiente crear propio):

```
class Persona:
    run = 1

    def __init__(self, nombre):
        self.nombre = nombre
        self.run = Persona.run
        Persona.run += 1

    def presentarse(self):
        print(f'Hola, soy {self.nombre} y mi run es {self.run}')

p1 = Persona('Hernán')
p2 = Persona('Fran')

p1.presentarse() # 1
p2.presentarse() # 2
```

Aprender más

- no.

Properties

Encapsulamiento

- Pendiente

Properties: property

- Crear getter, setter, y del rapidoo.

Porque es importante?

- Por temas de seguridad, permite checkear cosas antes de modificar un atributo directamente (por ejemplo que se le asigne un objeto de tipo correcto)
- mejorar esta parte

Forma simple de usarlas.

```
class Puente:

  def __init__(self, maximo):
    self.maximo = maximo
    self._personas = 0

    @property
    def personas(self): ## establece a cual variable apunta, y el getter.
      return self._personas

    @personas.setter
    def personas(self, p):
        if p > self.maximo:
            self._personas = self.maximo
        elif p < 0:
            self._personas = 0
        else:
            self._personas = p

## fuera de class
puente = Puente(10)
puente.personas += 11
puente.personas # es 10, ya que se checkea al setear, y se supera el maximo
## los checkeos se aplican a cualquier setting.
```

Ejemplos, añadir

- Otro ejemplo oficial

```
class Email2:

    def __init__(self, address):
        self._email = address

    @property
    def email(self):
        return self._email

    @email.setter
    def email(self, value):
        if '@' not in value:
            print("Esto no parece una dirección de correo.")
        else:
            self._email = value

    @email.deleter
    def email(self):
        print("¡Eliminaste el correo!")
        del self._email

mail = Email2("kp1@gmail.com")
print(mail.email)
mail.email = "kp2@gmail.com"
print(mail.email)
mail.email = "kp2.com"
del mail.email
```

Propuestos

Ej 2

```
class Misterio:
    def __init__(self, atributo):
        self._atributo = atributo

    @property
    def atributo(self):
        return self._atributo

    @atributo.setter
    def atributo(self, valor):
        if valor % 2 == 1:
            self._atributo = valor + 1
        else:
            self._atributo = valor
```

```
if __name__ == "__main__":
    misterio = Misterio(1)
    misterio.atributo += 1

## la solucion era que habia que usar
##self._atributo y no self.atributo
```

Ej 3

```
class DCCgram:
    def __init__(self):
        self._usuarios = []

    def agregar(self, nuevo_usuario):
        current_usernames = [usuario.username for usuario in self._usuarios]
        if not nuevo_usuario.username == None and \
            not nuevo_usuario.mail == None and \
            not nuevo_usuario.edad == None and \
            not nuevo_usuario.rut == None and \
            nuevo_usuario.username not in current_usernames:
            self.usuarios.append(nuevo_usuario)
        pass
    @property
    def usuarios(self):
        return self._usuarios
    @usuarios.setter
    def usuarios(self, valor):
        self._usuarios = valor

class Usuario:
    def __init__(self, username, mail, edad, rut):
        # Así validamos de inmediato los atributos
        self.__username = None
        self.__mail = None
        self.__edad = None
        self.__rut = None
        self.username = username
        self.mail = mail
        self.edad = edad
        self.rut = rut

    @property
    def username(self):
        return self.__username

    @username.setter
    def username(self, value):
        self.__username = value

    @property
    def mail(self):
```

```
        return self.__mail

    @mail.setter
    def mail(self, value):
        temp_list = value.split("@", 1)
        if len(temp_list) == 2 and temp_list[1] == "uc.cl":
            self.__mail = value

    @property
    def edad(self):
        return self.__edad

    @edad.setter
    def edad(self, value):
        if value >= 18:
            self.__edad = value

    @property
    def rut(self):
        return self.__rut

    @rut.setter
    def rut(self, value):
        #WIP: check digits##not wip anymore
        ct = 0
        for char in value:
            if char in '0123456789':
                ct += 1
        if(value.count('-') == 1 and ct <= 10):
            self.__rut = value
        #pass



if __name__ == "__main__":
    dcc_gram = DCCgram()
    u1 = Usuario('usuario1', 'usuario1@uc.cl', 17, '00000-0')
    u2 = Usuario('usuario2', 'usuario2@uc.cl', 19, '00000')
    u3 = Usuario('usuario3', 'usuario1@gmail.cl', 19, '00001-0')
    u4 = Usuario('usuario4', 'usuario4@uc.cl', 18, '00002-0')

    dcc_gram.agregar(u1)
    dcc_gram.agregar(u2)
    dcc_gram.agregar(u3)
    dcc_gram.agregar(u4)
    # Si todo ha salido bien, solo user 4 debería estar en la lista
    print(dcc_gram.usuarios)  # ¿Qué método deberias implementar para poder verlo en l
a lista?
    print(dcc_gram.usuarios[0].username)
```