



SECRETARÍA DE
INNOVACIÓN

Agenda

Sesión 10/18

1. Punto 1: creación de clases
2. Punto 2: métodos
3. Punto 3: atributos
4. Punto 4: herencia

PROGRAMACIÓN CON PYTHON

CREACIÓN DE CLASES

Clases

Sintaxis

```
class MiClase: #nomenclatura camel case  
    pass
```

#Creación de instancias (objetos)

```
nombre = MiClase() #instanciar, objeto
```

```
edad = MiClase() #no new como en otros lenguajes
```

#devuelve el tipo de clase del objeto

```
print(type(nombre))
```

PROGRAMACIÓN CON PYTHON

MÉTODOS

Métodos

Los objetos pueden realizar acciones. Las acciones se definen (construyen o realizan) utilizando métodos.

Un método es una función dentro de una clase.

```
class MiClase:
```

```
    #método (función)
```

```
    def funcionenclase(self): #self = "en si" por  
                                convención usamos self  
        print("Bienvenidos al curso de Python 3")
```

```
nombre = MiClase() #objeto
```

```
edad = MiClase() #objeto
```

```
nombre.funcionenclase()
```

```
edad.funcionenclase()
```

Métodos

```
class MiClase:
```

```
    def funcionenclase(self,elnombre): #agregamos parámetro  
        print(elnombre + " Bienvenido al curso de Python 3")
```

```
nombre = MiClase()
```

```
edad = MiClase()
```

```
nombre.funcionenclase("Francisco Quezada")
```

Métodos

```
class MiClase:
```

```
    def funcionenclase(self,elnombre):
```

```
        return elnombre + " Bienvenido al curso de Python 3"
```

```
nombre = MiClase()
```

```
print(nombre.funcionenclase("Francisco Quezada"))
```


PROGRAMACIÓN CON PYTHON

ATRIBUTOS

Atributos

Los objetos pueden tener cualquier cantidad de atributos.

Para nuestro ejemplo apellido, correo, dui, etc.

Podemos crear atributos fuera y dentro de la clase.

Atributos

```
class Alumnos:
```

```
    def fnalumnos(self,elnombre):
```

```
        return elnombre + " Bienvenido al curso de Python 3"
```

```
alumno = Alumnos()
```

```
#atributos
```

```
alumno.edad=33
```

```
alumno.nombre="Francisco"
```

```
alumno.apellido="Quezada"
```

```
alumno.dui="03569852-0"
```

```
alumno.email="alumno@alumno.com"
```

Atributos

```
class Alumnos:
```

```
    def fnalumnos(self,elnombre):
```

```
        return elnombre + " Bienvenido al curso de Python 3"
```

```
    def imprime_nombre(self):
```

```
        print(self.nombre)
```

```
        print(self.apellido)
```

```
alumno = Alumnos()
```

```
#atributos
```

```
alumno.edad=33
```

```
alumno.nombre="Francisco"
```

```
alumno.apellido="Quezada"
```

```
alumno.dui="03569852-0"
```

```
alumno.email="alumno@alumno.com"
```

```
alumno.imprime_nombre()
```

class Registro:

Atributos

def imprime_nombre(self):

#self hace referencia al objeto

print(self.nombre)

#alumno.nombre

#instructor.nombre

#atributos de los objetos fuera de la clase

alumno = Registro()

alumno.edad=30

alumno.nombre="Luis"

alumno.apellido="Castro"

alumno.dui="11111111-0"

alumno.email="alumno@alumno.com"

instructor = Registro()

instructor.edad=33

instructor.nombre="Francisco"

instructor.apellido="Quezada"

instructor.dui="03569852-0"

instructor.email="instructor@instructor.com"

alumno.imprime_nombre()

instructor.imprime_nombre()

#Creando atributos para los objetos dentro de la clase

class Registro:

def imprime_nombre(self):

print(self.nombre)

#método que crea atributo

def crea_objeto_nombre(self, nombre):

self.nombre = nombre #objeto.nombre=nombre

#método que imprime el atributo

def imprime_objeto_nombre(self):


print(self.nombre)

alumno = Registro()

alumno.crea_objeto_nombre("Francisco Quezada")

alumno.imprime_objeto_nombre()

Atributos



Debemos definir desde un inicio donde crearemos los atributos, si fuera o dentro de las clases, esto evitará confusión a la hora de buscar nuestros atributos en el código.

class Registro:

```
def __init__(self):
```

```
    self.nombre="
```

```
    self.apellido="
```

```
    self.edad="
```

```
    self.dui="
```

```
    self.sexo="
```

"""gracias a init nos ahorramos este tipo de creación de métodos:""

```
#def crea_objeto_nombre(self, nombre):
```

```
    #self.nombre = nombre
```

```
def imprime_objeto_nombre(self):
```

```
    print(self.nombre)
```

```
alumno = Registro()
```

#y este tipo de asignación

```
#alumno.edad=33
```

```
#alumno.nombre="Francisco"
```

```
#alumno.apellido="Quezada"
```

```
#alumno.dui="03569852-0"
```

```
#alumno.email="alumno@alumno.com"
```

```
alumno.crea_objeto_nombre("Francisco Quezada")
```

```
alumno.imprime_objeto_nombre()
```


class Registro:

def __init__(self):

self.nombre = "

self.apellido = "

self.edad = "

self.dui = "

self.sexo = "

def mensaje(self):

return "Bienvenido al curso de Python 3:" + self.nombre

def imprime_apellido(self):

print(self.apellido)

alumno = Registro()

respuesta = alumno.mensaje()

print(respuesta) #obtenemos el mensaje sin el nombre del alumno porque en línea 3 = "

class Registro:

```
def __init__(self, nombre="", apellido="", edad="", dui="", sexo="):
```

```
    self.nombre = nombre #valor por defecto
```

```
    self.apellido = apellido
```

```
    self.edad = edad
```

```
    self.oui = oui
```

```
    self.sexo = sexo
```

```
def mensaje(self):
```

```
    return "Bienvenido al curso de Python 3:" + self.nombre
```

```
def imprime_apellido(self):
```

```
    print(self.apellido)
```

```
alumno = Registro()
```

```
respuesta = alumno.mensaje()
```

```
print(respuesta) #obtenemos el mismo resultado
```

class Registro:

```
def __init__(self, nombre="", apellido="", edad="", dui="", sexo="): #constructor
```

```
    self.nombre = nombre
```

```
    self.apellido = apellido
```

```
    self.edad = edad
```

```
    self.oui = oui
```

```
    self.sexo = sexo
```

```
def mensaje(self):
```

```
    return "Bienvenido al curso de Python 3:" + self.nombre
```

```
def imprime_apellido(self):
```

```
    print(self.apellido)
```

#asignamos argumentos a nuestra instancia

```
alumno = Registro("Francisco","Quezada",33,"01256459-2","Hombre")
```

```
respuesta = alumno.mensaje()
```

```
print(respuesta)
```

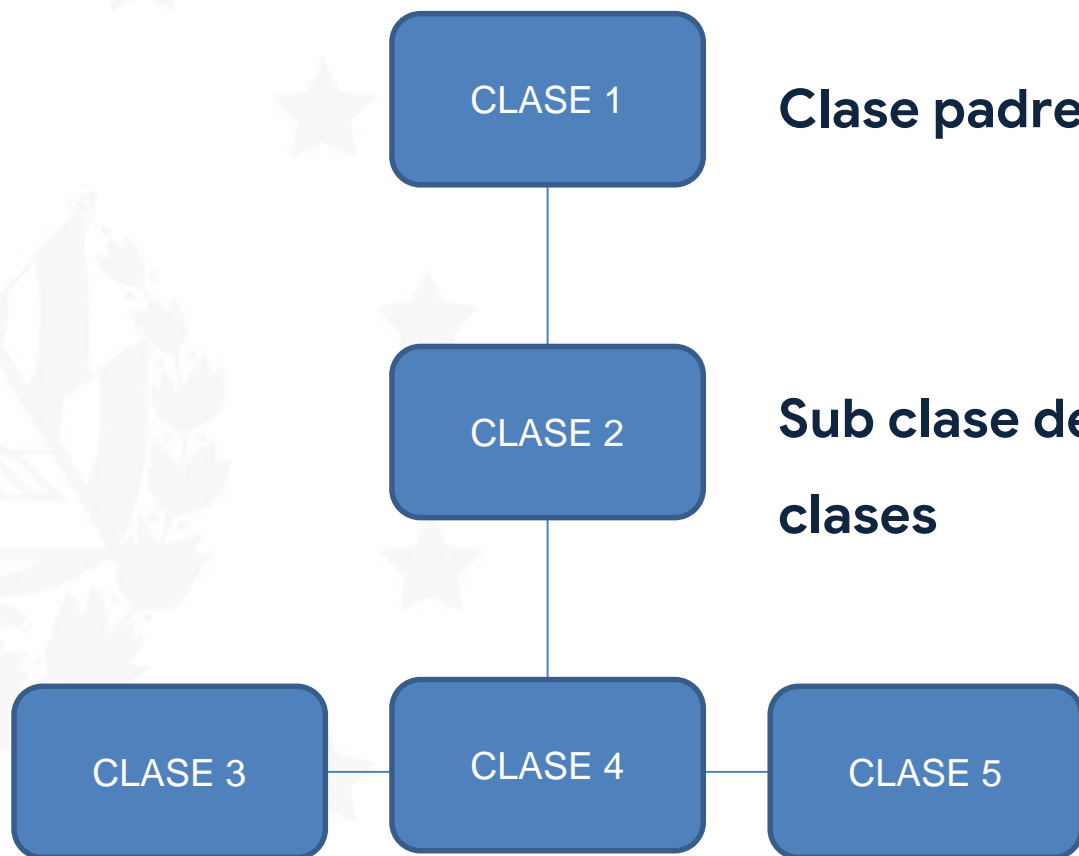
PROGRAMACIÓN CON PYTHON

HERENCIA

Herencia

En POO podemos crear clases a partir de otras clases.

Con esto se logra reducir significativamente nuestro código y reutilizarlo.



Clase padre / súper clase

Sub clase de la clase padre y súper clase del resto de clases

Herencia

¿Para qué nos sirve la herencia?

Reducir código.

Reutilización de código.

Optimización de procesos.

Herencia

Ejercicio:

- **Los vehículos de transporte son parte de una marca y un modelo.**
- **Los vehículos arrancan, aceleran y frenan.**
- **Desarrolle una clase que nos evite repetir el código necesario para todos los vehículos de transporte que tengan características y comportamientos en común.**

Herencia



MARCA

MODELO

ARRANCAN

ACELERAN

FRENAN

class Vehiculos:

#constructor para dar estado inicial

def __init__(self, marca, modelo):

#propiedades

#marca será igual a la marca que pasemos por parámetro

self.marca=marca

self.modelo=modelo

self.enmovimiento=False #no está en movimiento

self.acelera=False

self.frena=False

'''

**Quando creamos un objeto vehiculo
el objeto deberá**

**tener una marca y un modelo que
deberemos mandar**

**al constructor. Lo mismo sucederá
si creamos otro**

objeto que herede del vehiculo.

'''

class Vehiculos:

def __init__(self, marca, modelo):

self.marca=marca

self.modelo=modelo

self.enmovimiento=False

self.acelera=False

self.frena=False

#comportamientos (métodos) del objeto vehiculo

def semueve(self):

self.enmovimiento=True

def acelerar(self):

self.acelera=True

def frena(self):

self.frena=True

def estado_vehiculo(self):

print("Marca:", "Modelo:", "En movimiento:", "Acelerando: ", "Frenando: ")

#ejecutamos

#agregamos el siguiente código

#creando objetos

#moto que hereda de la clase vehiculos (propiedades y métodos)

#sintaxis para heredar

class Moto(Vehiculos): #en () nombre de la clase que heredamos
pass

#creamos las instancias de la clase moto

#podremos utilizar los métodos de la clase vehiculos

lamoto=Moto("Toyota", "Corolla") #parámetros marca y modelo

#llamamos a cualquiera de los métodos heredados

lamoto.estado_vehiculo()

#Mejoremos el código

```
print("Marca:", self.marca, "Modelo:", self.modelo, "En movimiento:", self.enmovimiento,  
"Acelerando: ", self.acelera, "Frenando: ", self.frena)
```

#Mejóremelo más

```
print("Marca:", self.marca, "\nModelo:", self.modelo, "\nEn movimiento:", self.enmovimiento,  
"\nAcelerando: ", self.acelera, "\nFrenando: ", self.frena)
```

RESUMEN DE SESIÓN



SECRETARÍA DE
INNOVACIÓN