title: Presentacion Python 2019
Author: Claudia Banchoff, Viviana Harari
description: clase 8
keywords: POO
css: estilo.css

Seminario de Lenguajes - Python

Cursada 2019

Temario

Programación Orientada a Objetos en Python

(Parte I)

Consigna

- Retomemos el trabajo que tenían que modificar sobre los juegos. La consigna era registrar los datos de quien jugaba en un archivo.
- Entonces, podemos pensar en una entidad "Jugador" con datos asociados tales como:
    - Nombre
    - Nick
    - Contraseña
    - Nivel
    - Puntaje acumulado
    - Vidas restantes
- ¿Con qué estructura de datos trabajaron?

Con lo que sabemos hasta ahora...

- ... podríamos utilizar **diccionarios** para definir su estructura:

```
jugador = {'nombre': 'Tony', 'nick': 'Ironman', 'contraseña': '123456NOOOO!', ...}
```

- ¿Cómo puedo asociar alguna funcionalidad específica a un "jugador"? Por ejemplo, **incrementar su puntaje**, **incrementar/decrementar sus vidas**, etc.

Con lo que sabemos hasta ahora...

- ... podríamos usar **funciones** para definir la funcionalidad asociada:

```
def incrementar_vidas(jugador, cant_vidas):
        jugador["vidas"] += cant_vidas
```

Pero...

¿Cómo hacemos si queremos...

- ... definir un "jugador" como una **entidad** que **encapsule** tanto la estructura como la funcionalidad para manipularla?

Hablemos de objetos

---

class: destacado

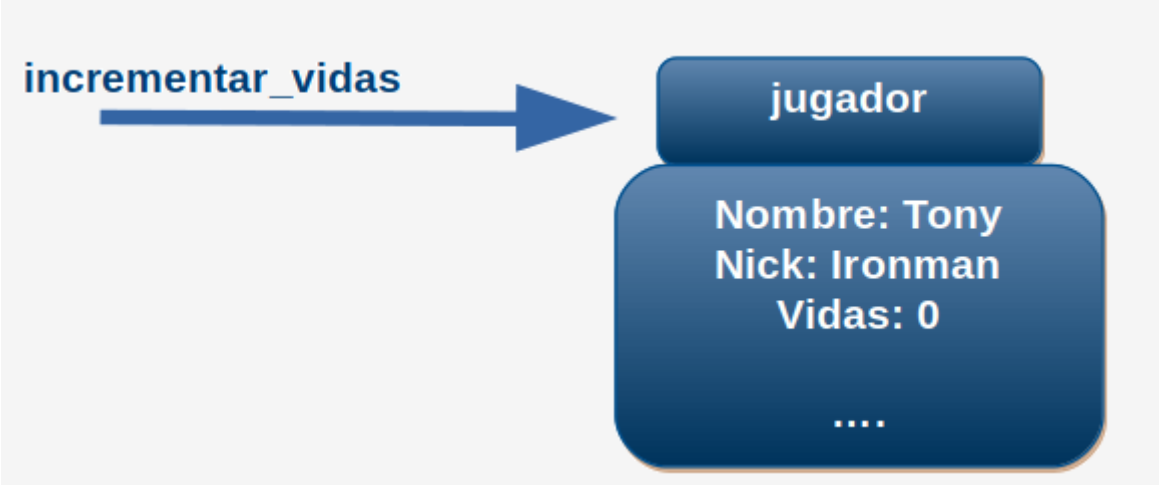
POO: conceptos básicos

Un objeto es una colección de datos con un comportamiento asociado.

En nuestro ejemplo...

POO: conceptos básicos

- En POO un programa puede verse como un **conjunto de objetos** que interactúan entre ellos **enviándose mensajes**.
- Estos mensajes están asociados al comportamiento del objeto.



POO: conceptos básicos

Algunos términos con los que vamos a trabajar:

- Objetos
- Mensajes
- Métodos
- Clases

Objetos

- Son los elementos fundamentales de la POO.
- Son entidades que poseen **estado interno** y **comportamiento**.
- Ya vimos anteriormente, que muchos de los elementos con los que trabajamos son objetos.
- Ejemplos:

```
cadena = "Hola"
archivo = open("archi.txt")

cadena.upper()
archivo.close()
```

- **cadena** y **archivo** son objetos.
- **upper** y **close** forman parte del comportamiento de estos objetos: son **métodos**.

---

class: destacado

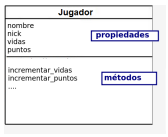
Objetos y clases

- No todos los objetos son iguales, ni tienen el mismo comportamiento.
- Así agrupamos a los objetos de acuerdo a características comunes.

Una clase describe las **propiedades** o atributos de objetos y las acciones o **métodos** que pueden hacer o ejecutar dichos objetos.

Clases

- Ejemplo: clase **Jugador**



- Cuando creamos un objeto, creamos una **instancia de la clase**.
- Una instancia es un objeto individualizado por los valores que tomen sus atributos o propiedades.
- La **interfaz pública** del objeto está formada por las propiedades y métodos que otros objetos pueden usar para interactuar con él.
- ¿Qué pasa si todas las propiedades y métodos son privadas?
- ¿Y si son todas públicas?

Clases en Python

```
class NombreClase:
    sentencias
    ....
    sentencias
```

Usamos CamelCase para el nombre

- Al igual que las funciones, las clases **deben** estar definidas antes de que se utilicen.
- Con la definición de una nueva clase se crea un **espacio de nombres nuevo**.
- La siguiente sentencia crea una instancia de la clase:

```
objeto = NombreClase()
```

La clase Jugador

```
class Jugador():
        "Define la entidad que representa a un jugador en el juego"

        #Propiedades
        nombre = 'Tony'
        nick = 'Ironman'
        contraseña ='123456Nooooo!'
        contacto = 'tony.stark@gmail.com'
        vidas = 0
        puntaje = 0

        #Métodos
        def incrementar_vidas(self, cant_vidas):
                self.vidas += cant_vidas

jugador1 = Jugador()
jugador1.incrementar_vidas(10)
```

- ¿self?
- **jugador1.incrementar_vidas()**
- Cuando creamos otros objetos de clase **Jugador**, ¿qué particularidad tendrán?

El método __init()__

```
class Jugador():
        "Define la entidad que representa a un jugador en el juego"
        def __init__(self, nom, nic, clave, e_mail):
                self.nombre = nom
                self.nick = nic
                self.contraseña = clave
                self.contacto = e_mail
                self.vidas = 0
                self.puntaje = 0
                ........
        def incrementar_puntaje(self, cant_puntos):
                self.puntaje += cant_puntos

        def incrementar_vidas(self, cant_vidas):
                self.vidas += cant_vidas

jugador1 = Jugador('Tony','Ironman', '12345NOOOO', 'tony.stark@gmail.com')
jugador1.incrementar_vidas(10)
```

- El método **__init__() se invoca automáticamente** al crear el objeto.
- ¿Qué pasa si..?

```
jugador2 = Jugador()
```

Podemos inicializar con valores por defecto

```
class Jugador():
        "Define la entidad que representa a un jugador en el juego"
        def __init__(self, nom="Tony", nic="Ironman", clave="123", e_mail=""):
                self.nombre = nom
                self.nick = nic
                self.contraseña = clave
                self.mail = e_mail
                self.vidas = 0
                self.puntaje = 0
                ........
        def incrementar_puntaje(self, cant_puntos):
                self.puntaje += cant_puntos

        def incrementar_vidas(self, cant_vidas):
                self.vidas += cant_vidas

jugador1 = Jugador()
jugador2 = Jugador("Bruce", "Batman", "4321", "batimail@gmail.com")
print(jugador1.nombre)
print(jugador2.nombre)
```

- Observemos el código: ¿no hay algo que parece incorrecto?

---

class: destacado

Público y privado

- Antes de empezar a hablar de esto ....

"*"Private" instance variables that cannot be accessed except from inside an object don't exist in Python.*"

- Más info: https://docs.python.org/3/tutorial/classes.html#private-variables

¿Entonces?

---

class: destacado

Público y privado

- Hay una convención ..

Es posible **definir el acceso** a determinados métodos y atributos de los objetos, quedando claro qué cosas se pueden y no se pueden utilizar desde **fuera de la clase**.

¿Privado?

- Por convención, todo atributo (propiedad o método) que comienza con "_" se considera no público.
- Pero esto no impide que se acceda. **Simplemente es una convención**.

```
class Jugador():
        "Define la entidad que representa a un jugador en el juego"
        def __init__(self, nom="Tony", nic="Ironman", clave="123", e_mail=""):
                self._nombre = nom
                self.nick = nic
                self.contraseña = clave
                self.mail = e_mail
                self.vidas = 0
                self.puntaje = 0
        def incrementar_puntaje(self, cant_puntos):
                self.puntaje += cant_puntos

        def incrementar_vidas(self, cant_vidas):
                self.vidas += cant_vidas

jugador1 = Jugador()
print(jugador1._nombre)
```

class: destacado

¿Privado?

- Hay otra forma de indicar que algo no es "tan" público: agregando a los nombres de la variables o funciones, dos guiones **(__)** delante.

Veamos un ejemplo ....

Códigos secretos

```
class CodigoSecreto:
        '''¿¿¿   Textos con clave ???? '''

        def __init__(self, texto_plano, clave_secreta):
                self.__texto_plano = texto_plano
                self.__clave_secreta = clave_secreta

        def desencriptar(self, clave_secreta):
                '''Solo se muestra el texto si la clave es correcta'''
                if clave_secreta == self.__clave_secreta:
                        return self.__texto_plano
                else:
                        return ''
```

- Observemos el código:
    - ¿Cuáles son las propiedades? ¿Públicas o privadas?
    - ¿Y los métodos?¿Públicos o privados?
    - ¿Cómo creo un objeto **CodigoSecreto**?

Códigos secretos

```
class CodigoSecreto:
        '''¿¿¿   Textos con clave ???? '''

        def __init__(self, texto_plano, clave_secreta):
                self.__texto_plano = texto_plano
                self.__clave_secreta = clave_secreta

        def desencriptar(self, clave_secreta):
                '''Solo se muestra el texto si la clave es correcta'''
                if clave_secreta == self.__clave_secreta:
                        return self.__texto_plano
                else:
                        return ''

texto_secreto = CodigoSecreto("Seminario Python", "stark")
print(texto_secreto.desencriptar("stark"))
```

- ¿Qué pasa si tenemos quiero imprimir desde fuera de la clase: **texto_secreto.__texto_plano**?

Entonces, ¿si es privado?

---

class: destacado

Códigos no tan secretos

- Ahora, probemos esto:

```
print(texto_secreto._CodigoSecreto__texto_plano)
```

Todo identificador que comienza con **"__"**, por ejemplo **__texto_plano**, es reemplazado textualmente por _NombreClase__identificador, por ejemplo: **_CodigoSecreto__texto_plano**.

- +Info: https://dbader.org/blog/meaning-of-underscores-in-python

---

class: destacado

Público y privado

Respetaremos las convenciones y todo identificador que comienza con "__" será considerado privado, aunque sea técnicamente posible acceder a ese elemento fuera de la clase.

Retomemos la clase Jugador

- Analicemos qué información debería ser privada y cuál pública.

```
class Jugador ():
        "Define la entidad que representa a un jugador en el juego"
        def __init__(self, nom="Tony", nic="Ironman", clave="123", e_mail=""):
                self.__nombre = nom
                self.__nick = nic
                self.__contraseña = clave
                self.__mail = e_mail
                self.__vidas = 0
                self.__puntaje = 0
        def incrementar_puntaje(self, cant_puntos):
                self.__puntaje += cant_puntos

        def incrementar_vidas(self, cant_vidas):
                self.__vidas += cant_vidas


jugador1 = Jugador()
jugador2 = Jugador("Bruce", "Batman", "4321", "batimail@gmail.com")

print (jugador1.__nick)
```

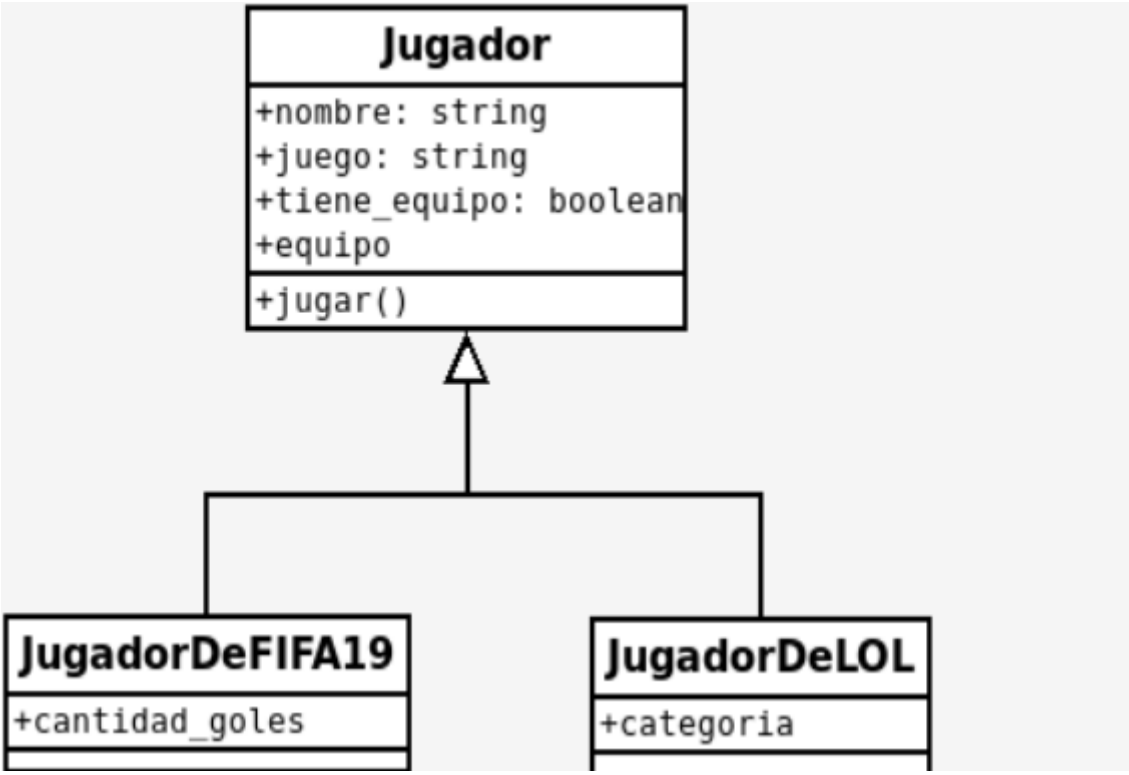
- ¿Tiene sentido definir incrementar_puntaje e incrementar_vidas como privados?

Algunos métodos especiales

- __str__
- __lt__, __gt__, __le__, __ge__
- __eq__, __ne__
- Veamos este ejemplo: metodos_especiales.py

Juegos electrónicos

- Observemos estas clases



- Un jugador de LOL "es un" Jugador.

Herencia

- Es uno de los conceptos más importantes de la POO.
- La herencia permite que una clase pueda *heredar* los atributos y métodos de otra clase, a parte de que ella tengas sus atributos y métodos propios.
- Con herencia se suma, se extiende una clase.
- La clase que hereda se denomina **clase derivada** y la clase de la cual se deriva se denomina **clase base**.
- Así, Jugador es la clase base y JugadorDeLOL es la clase derivada.

¿Escribimos el código?

```
class Jugador:
        def __init__(self, nombre, juego="Tetris", tiene_equipo= False, equipo=None):
                self.nombre = nombre
                self.juego = juego
                self.tiene_equipo = tiene_equipo
                self.equipo = equipo

        def jugar(self):
                if self.tiene_equipo:
                        print ("{} juega en equipo al {}".format(self.nombre, self.juego
                else:
                        print("{} juega solo al {}".format(self.nombre, self.juego))
class JugadorDeFIFA19(Jugador):
        def __init__(self, nombre):
                Jugador.__init__(self, nombre, "PS4")
class JugadorDeLOL(Jugador):
        def __init__(self, nombre, equipo):
                Jugador.__init__(self, nombre, "LOL")

nico = JugadorDeFIFA19('Nico Villalba')
nico.jugar()
faker = JugadorDeLOL("Faker", "SK Telecom")
faker.jugar()
```

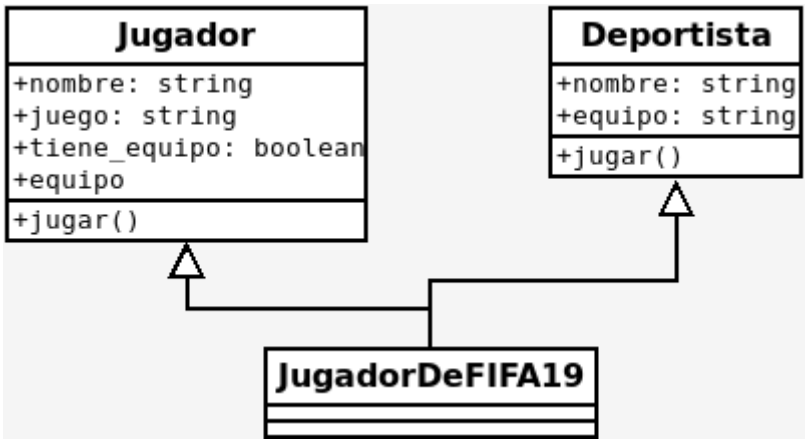
- La función **super()**

---

class: destacado

¿e-sports?

- Observemos estas clases



- Un jugador de FIFA19 "es un" Jugador, pero también "es un" Deportista.

Python tiene **herencia múltiple**

¿A que jugamos?

- Veamos el ejemplo: herencia_multiple.py
- Ambas clases bases tienen definido un método **jugar**
    - En este caso, se toma el método de la clase más a la **izquierda** de la lista.
- Por lo tanto, es MUY importante el orden en que se especifican las clases bases.

Resumiendo...

---

class: destacado

Objetos y clases

- La **clase** define las propiedades y los métodos de los objetos.
- Los **objetos** son instancias de una clase.
- Cuando se crea un objeto, se ejecuta el método __**init()**__ que permite inicializar el objeto.
- La definición de la clase especifica qué partes son privadas y cuáles públicas.

---

class: destacado

Mensajes y métodos

TODO el procesamiento en este modelo es activado por mensajes entre objetos.

- El **mensaje** es el modo de comunicación entre los objetos. Cuando se invoca una función de un objeto, lo que se está haciendo es **enviando un mensaje** a dicho objeto.
- El **método** es la función que está asociada a un objeto determinado y cuya ejecución sólo puede desencadenarse a través del envío de un mensaje recibido.

Conceptos asociados a la POO

- Encapsulamiento
    - **class**, métodos privados y públicos, propiedades.
- Herencia
    - Clases bases y derivadas.
    - Herencia múltiple
- ¿Polimorfismo?

Polimorfismo

- Capacidad de los objetos de distintas clases de responder a mensajes con el mismo nombre.
- Ejemplo: + entre enteros y cadenas.
- En Python al no ser necesario especificar explícitamente el tipo de los parámetros que recibe una función, las funciones son naturalmente polimórficas.
- ¿Qué pasa con el siguiente código?

```
def saludo():
    print("¡Hola, mundo!")
def saludo():
    print("¡Hello, world!")
saludo()
```

- Ya vimos en el ejemplo de e-sport.

Seguimos en la próxima ...