ESCUELA POLITÉCNICA NACIONAL

cec·epn CENTRO DE EDUCACIÓN CONTINUA

UNIDAD 6

Programación orientada a objetos, manejo de archivos y librerias.

Manejo de archivos - Apertura

Python tiene funciones integradas para crear y manipular archivos, ya sea archivos planos o archivos de texto. IO es el módulo predeterminado para acceder a los archivos, por lo que no es necesario importar ninguna biblioteca externa para las operaciones generales de IO.

Las funciones clave utilizadas para el manejo de archivos en Python son: open(), close(), read().

Modo	Descripción
'r'	Este es el modo por default y es para leer
'w'	Este modo abre un archivo y escribe. Si el archivo no existe este crea un nuevo archivo. Si existe el archivo detiene la ejecución.
'x'	Crea un nuevo archivo, si existe falla la operación
'a'	Abrir archivo en modo agregar. Si no existe el archivo crea uno nuevo.
't'	Este es por default abre el modo de texto
'b'	Abre en modo binario
'+'	Esto abrirá un archivo para leer y escribir (Actualizara)

```
1.  # Abre el archivo para escribir y elimina los archivos anteriores si existen
2.  fic = open("text.txt", "w")
3.
4.  # Abre el archivo para agregar contenido
5.  fic = open("text.txt", "a")
6.
7.  # Abre el archivo en modo lectura
8.  fic = open("text.txt", "r")
```

Escritura

Para la escritura de archivos se utilizan los métodos write.

```
archi1.write("Primer línea.\n")
archi1.write("Segunda línea.\n")
```

Lectura

El método read devuelve una cadena con el contenido del archivo:

```
#Abrir archivo en modo escritura
fw = open("archivo.txt", "w")
#Contenido completo
completo = fr.read()
```

El método readline sirve para leer las líneas del fichero una por una.

```
while True:
    linea = fr.readline()
    if not linea:
        break
    print(linea)
```

El método readlines funciona leyendo todas las líneas del archivo y devolviendo una lista con las líneas leídas.

Para leer los caracteres especiales correctamente, puede pasar el parámetro de codificación a read() función y establezca su valor en utf8:

```
example = open("C:\\Users\\Documents\\ejemplo.txt","r", encoding="utf8")
print(example.read())
```

Cerrar archivos

La función close() borra el búfer de memoria y cierra el archivo. Esto significa que ya no se podrá leer del archivo. Algunos sistemas operativos, como Windows, tratan los archivos abiertos como bloqueados, por lo que es importante cerrar un archivo después de usarlo.

```
file = open("demo.txt", "w")
file.write("This is a test")
file.write("To add more lines.")
file.close()
```

Módulos y librerías.

- De forma general un módulo es una porción de código que realiza una o varias tareas.
- Una biblioteca o librería es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.

  La orden from (module) import permite importar determinadas funciones de un módulo o el modulo completo para no referenciar el nombre de los objetos a su módulo.

```
5  from math import *
6  x = int(input("Ingresa un numero \n"))
7  raiz = sqrt(x)
8  print (raiz)
```

La instrucción print (dir(modulo)) permite obtener una lista de las funciones y métodos de cada módulo o libreria.

Objetos - programación orientada a objetos

En Python todo es un objeto. Al crear una variable asignarle un valor entero, ese valor es un objeto; una función es un objeto; las listas, tuplas, diccionarios, son objetos; una cadena de caracteres es un objeto.

Clases - programación orientada a objetos

De forma muy general, una clase es una plantilla o modelo para crear a partir de ella ciertos objetos. Esta plantilla es la que contiene la información; características y capacidades que tendrá el objeto que sea creado a partir de ella.

Así a su vez los objetos creados a partir de ella estarán agrupados en esa misma clase.

El concepto de la programación orientada a objetos consiste en resolver problemas grandes subdividiendo un programa en otros más pequeños a cargo de objetos con determinadas características y tareas encomendadas.

La creación de un objeto a partir de una clase se denomina instanciar.

Atributos y métodos de un objeto.

Los objetos almacenan información y realizan tareas. Estos atributos pueden llamarse las características o propiedades que se definen para ese objeto. Y los métodos las tareas que son capaces de realizar.

Ejemplo:

Se tiene la Clase: Animal

El objeto instanciado (creado a partir de esa clase): Perro

Tiene sus atributos:

Color: negro;

Raza: french;

Y tiene sus métodos:

Ladra (“Guau; guau”)

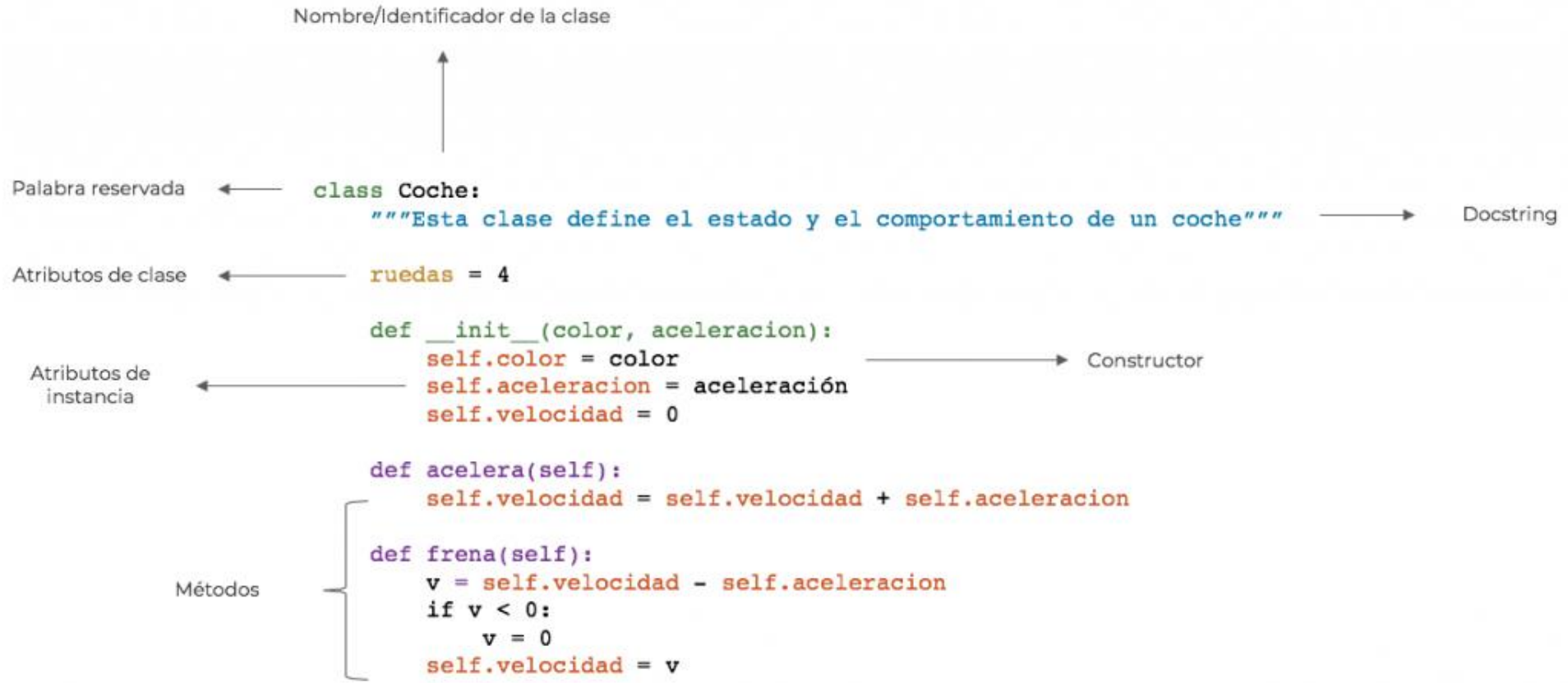
Camina()

Los atributos y métodos se definen al crear la clase y luego al momento de instanciar se le brindan los “argumentos” para esos atributos y métodos.

Los atributos del objeto normalmente se almacenan en variables declaradas en la clase.

Y los métodos son funciones que el objeto puede realizar y que puede necesitar de argumentos para trabajar con sus parámetros.

Sintaxis



Para crear una clase se utiliza la expresión class seguida del nombre de la clase,(entre paréntesis la clase de la cual puede heredar, una clase puede recibir atributos y métodos de otra - herencia).

Al crear una clase si este parámetro entre paréntesis no se declara; la clase automáticamente heredara de Object (Es una clase predefinida en el propio lenguaje). Dentro del bloque de código se define el método constructor de la clase.

Class Nombredelaclase (object): #Declaración de la clase
def __init__ (self, parámetros): #Constructor de la clase

#Declaración de atributos

Self no es en realidad una palabra reservada del lenguaje. Esta hace referencia al objeto en instancia pero no obligadamente tiene que tener ese nombre.

En Python es posible usar cualquier nombre en lugar de Self; aunque es buena practica simplemente usar self para referirse al mismo objeto.

Clase y objeto - Instanciar:

```
class Humano(): #Creamos la clase Humano
    def __init__(self, edad, nombre): #Definimos el atributo edad y nombre
        self.edad = edad # Definimos que el atributo edad, sera la edad asignada
        self.nombre = nombre # Definimos que el atributo nombre, sera el nombre asig

Persona1 = Humano(31, "Pedro") #Instancia
```

Añadir métodos y atributos

```
class Humano(): #Creamos la clase Humano
    def __init__(self, edad, nombre, ocupacion): #Definimos el parámetro edad , nombre y ocupacion
        self.edad = edad # Definimos que el atributo edad, sera la edad asignada
        self.nombre = nombre # Definimos que el atributo nombre, sera el nombre asig
        self.ocupacion = ocupacion #DEFINIMOS EL ATRIBUTO DE INSTANCIA OCUPACION

Persona1 = Humano(31, "Pedro", "Desocupado") #Instancia
```

```
class Humano(): #Creamos la clase Humano
    def __init__(self, edad, nombre, ocupacion): #Definimos el parametro edad , nombre y ocupacion
        self.edad = edad # Definimos que el atributo edad, sera la edad asignada
        self.nombre = nombre # Definimos que el atributo nombre, sera el nombre asig
        self.ocupacion = ocupacion #DEFINIMOS EL ATRIBUTO DE INSTANCIA OCUPACION

        #Creación de un nuevo método
    def presentar(self):
        presentacion = ("Hola soy {}, mi edad es {} y mi ocupación es {}") #Mensaje


Persona1 = Humano(31, "Pedro", "Desocupado") #Instancia

#Llamamos al método

Persona1.presentar()
```

Es posible manipular atributos mediante métodos internos de la clase.

```
class Humano(): #Creamos la clase Humano
    def __init__(self, edad, nombre, ocupacion): #Definimos el parámetro edad , nombre y ocupación
        self.edad = edad # Definimos que el atributo edad, sera la edad asignada
        self.nombre = nombre # Definimos que el atributo nombre, sera el nombre asig
        self.ocupacion = ocupacion #DEFINIMOS EL ATRIBUTO DE INSTANCIA OCUPACIÓN

        #Creación de un nuevo método
    def presentar(self):
        presentacion = ("Hola soy {}, mi edad es {} y mi ocupación es {}") #Mensaje
        print(presentacion.format(self.nombre, self.edad, self.ocupacion)) #Usamos FORMAT


        #Creamos un nuevo método para cambiar la ocupación:
        #En caso que esta persona sea contratada

    def contratar(self, puesto): #añadimos un nuevo parámetro en el método
        self.puesto = puesto
        print ("{} ha sido contratado como {}"
        self.ocupacion = puesto #Ahora cambiamos el atributo ocupación
Persona1 = Humano(31, "Pedro", "Desocupado") #Instancia

#Llamamos al método

Persona1.presentar()
Persona1.contratar("Obrero")
Persona1.presentar() #Lo volvemos a presentar luego de su contratación
```

Herencia

Es una técnica de programación orientada a objetos que permite crear una clase general primero y luego más tarde crear clases más especializadas que re-utilicen código de la clase general.

La herencia también le permite escribir un código más limpio y legible.

Se crea una clase hija a partir de una clase padre. La clase hija hereda todo (miembros de datos y funciones de miembros) de la clase padre.

```
class ParentClass:
    #members
class ChildClass(ParentClass):
    #members
```

En la clase hija, se tienen todas las características de la clase padre y también se pueden añadir nuevas funcionalidades.

```
>>> class Auto:
    def __init__(self, e, n):
        self.engine = e
        self.name = n
    def display(self):
        print("Name of Auto: ", self.name)
        print("Engine of auto: ", self.engine)


>>> class Car(Auto):
    def __init__(self):
        self.x= input("Enter name of car: ")
        self.y= input("Enter engine of car: ")
        Auto.__init__(self,self.y,self.x)


>>> c = Car()
Enter name of car: Prius
Enter engine of car: 1.5l
>>> c.display()
Name of Auto:  Prius
Engine of auto:  1.5l
```


ESCUELA
POLITÉCNICA
NACIONAL


cec·epn
CENTRO DE EDUCACIÓN CONTINUA


python™