

Temas

- El enfoque orientado a objetos POO
- Fundamentos
- Objetos
- Clases
- Trabajando con archivos

Programación Orientada a Objetos en Python

POO en Python

CEC-EPN

Juan Carlos Domínguez Ayala

02/2022



Introducción

La Programación Orientada a Objetos (POO u OOP por sus siglas en inglés), es un paradigma de programación.

Paradigma: Teoría cuyo núcleo central suministra la base y modelo para resolver problemas

“Definición de la Real Academia Española”

Descripción general de la terminología de POO

Clase :

- Un prototipo definido por el usuario para un objeto que define un conjunto de atributos que caracterizan cualquier objeto de la clase.
- Los atributos son miembros de datos (variables de clase y variables de instancia) y métodos, a los que se accede mediante notación de puntos.

Variable de clase :

- Una variable que comparten todas las instancias de una clase.
- Las variables de clase se definen dentro de una clase pero fuera de cualquiera de los métodos de la clase.
- Las variables de clase no se usan con tanta frecuencia como las variables de instancia.

Descripción general de la terminología de POO

Miembro de datos:

- Una variable de clase o variable de instancia que contiene datos asociados con una clase y sus objetos.

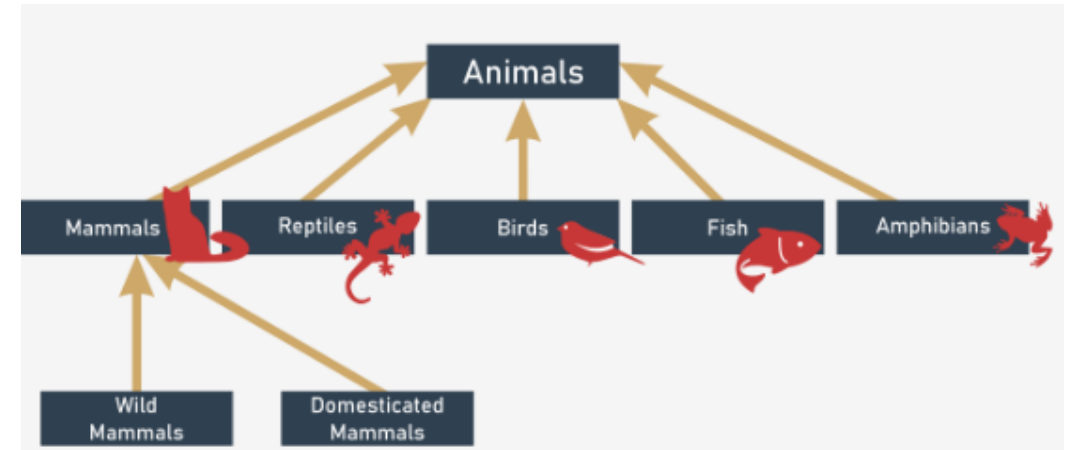
Sobrecarga de funciones:

- La asignación de más de un comportamiento a una función en particular.
- La operación realizada varía según los tipos de objetos o argumentos involucrados.

Variable de instancia :

- Una variable que se define dentro de un método y pertenece solo a la instancia actual de una clase.

Descripción general de la terminología de POO



Herencia :

La transferencia de las características de una clase a otras clases que se derivan de ella.

Instancia :

Un objeto individual de una determinada clase.

Un objeto obj que pertenece a una clase Circle, por ejemplo, es una instancia de la clase Circle.

Descripción general de la terminología de POO

Instanciación : La creación de una instancia de una clase.

Método : Un tipo especial de función que se define en una definición de clase.

Objeto :

- Una instancia única de una estructura de datos definida por su clase.
- Un objeto comprende miembros de datos (variables de clase y variables de instancia) y métodos.

Sobrecarga del operador : La asignación de más de una función a un operador en particular.

Introducción

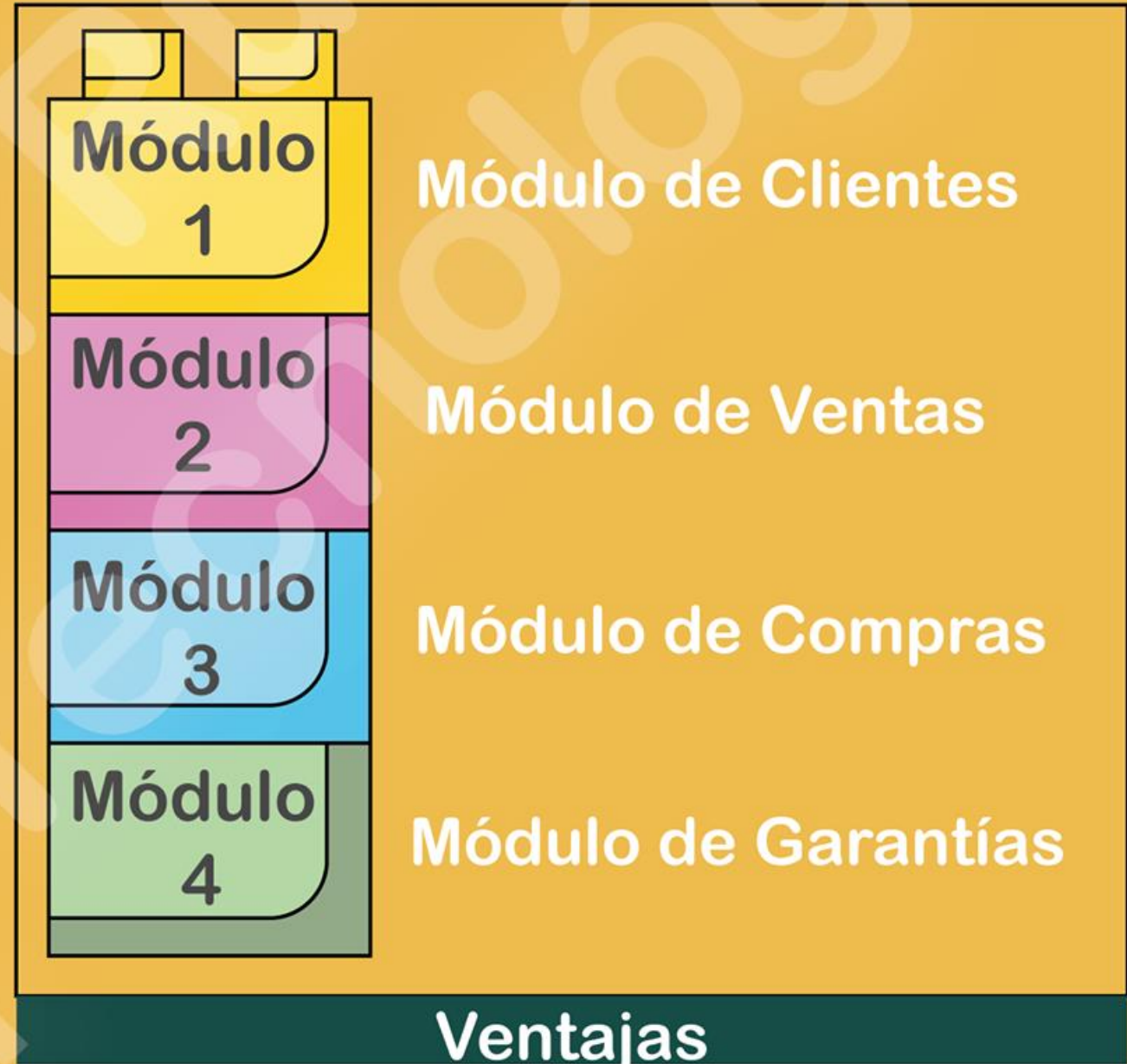
Un Programa Orientado a Objetos (POO) se basa en una agrupación de objetos de distintas clases que interactúan entre sí y que, en conjunto, consiguen que un programa cumpla su propósito.

En este paradigma de programación se intenta emular el funcionamiento de los objetos que nos rodean en la vida real.

En Python cualquier elemento del lenguaje pertenece a una clase y todas las clases tienen el mismo rango y se utilizan del mismo modo.

Sistema de Ventas

Forma Tradicional



Pensar en objetos

- Pensar en objetos, puede resultar -al inicio- una tarea difícil.
- Sin embargo, difícil no significa complejo. Por el contrario, pensar en objetos representa la mayor simplicidad que uno podría esperar del mundo de la programación.
- Un objeto es “una cosa”. Y, si una cosa es un sustantivo, entonces un objeto es un sustantivo.
- Mira a tu alrededor y encontrarás decenas, cientos de objetos.
- Tu ordenador, es un objeto, Tú, eres un objeto, tu llave es un objeto, tu mascota también es un objeto.
- Cuando pensamos en “objetos”, todos los sustantivos son objetos.
- Sencillo ¿cierto? Entonces, de ahora en más, solo concéntrate en pensar la vida en objetos (al menos, hasta terminar de leer este documento).

Descripción de objetos

- Describir un objeto, es simplemente mencionar sus cualidades.
- Las cualidades son adjetivos.
- Un adjetivo es una cualidad del sustantivo.
- Entonces, para describir “la manera de ser” de un objeto, debemos preguntarnos ¿cómo es el objeto?
- Toda respuesta que comience por “el objeto es”, seguida de un adjetivo, será una cualidad del objeto.

Algunos ejemplos:

- El objeto es verde
- El objeto es grande

¿Qué tiene un objeto?

¿Qué tiene un objeto?

- La convención de programación de objetos supone que cada objeto existente puede estar equipado con tres grupos de atributos :
 1. Un objeto tiene un nombre que lo identifica de forma exclusiva dentro de su espacio de nombres de inicio
 2. Un objeto tiene un conjunto de propiedades individuales que lo hacen original, único o sobresaliente
 3. Un objeto tiene un conjunto de habilidades para realizar actividades específicas , capaz de cambiar el objeto mismo o algunos de los otros objetos.

¿Qué tiene un objeto?

- Pista que puede ayudar a identificar cualquiera de las tres atributos anteriores.
- Cada vez que describe un objeto y usa:
 1. Un sustantivo: probablemente defina el nombre del objeto;
 2. Un adjetivo: probablemente defina la propiedad del objeto;
 3. Un verbo: probablemente definas la actividad del objeto.

Ejemplos

Max es un gato grande que duerme todo el día.

- Nombre del objeto = Max
- Clase = gato
- Propiedad del gato = tamaño (grande)
- Actividad = Dormir (todo el día)

Ejemplos

- Un Cadillac rosado se fue rápidamente.
- Nombre del objeto = Cadillac
- Clase = Vehículos con ruedas
- Propiedad = Color (rosa)
- Actividad = Ir (rápidamente)

Max is a large cat who
sleeps all day



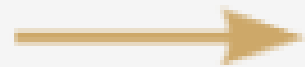
Home
Class



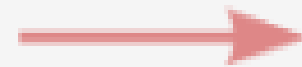
Cat

Object

name

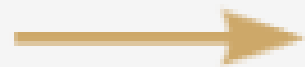


Noun



Max

properties

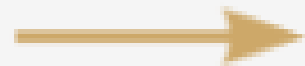


Adjective

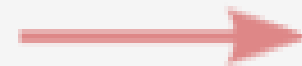


Size (Large)

activities



Verb

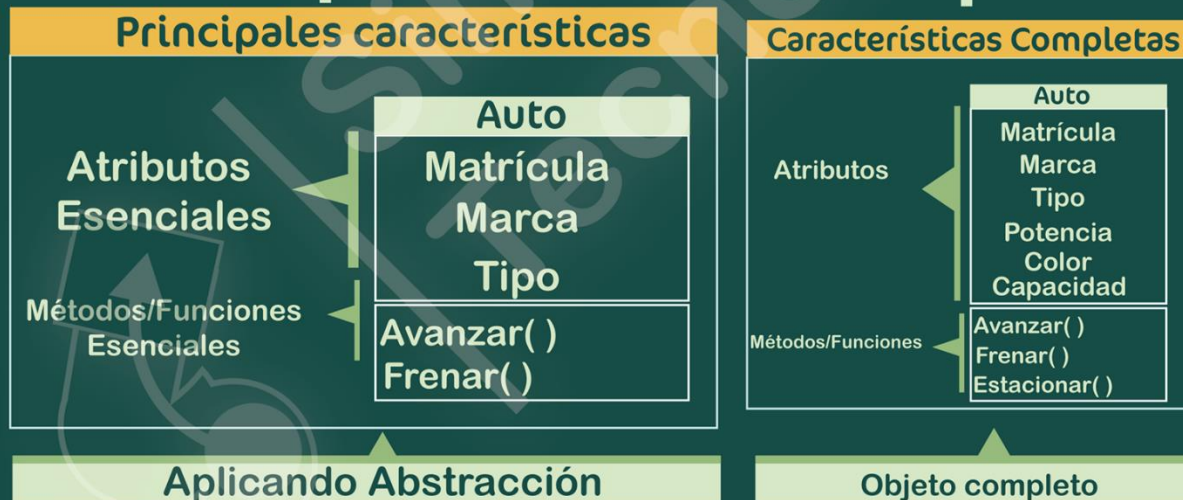


Sleep (all day)

Abstracción

Consiste en **describir un objeto con sus principales características que lo hacen distinguir del resto**, sin pensar mucho en los detalles

Objeto Auto



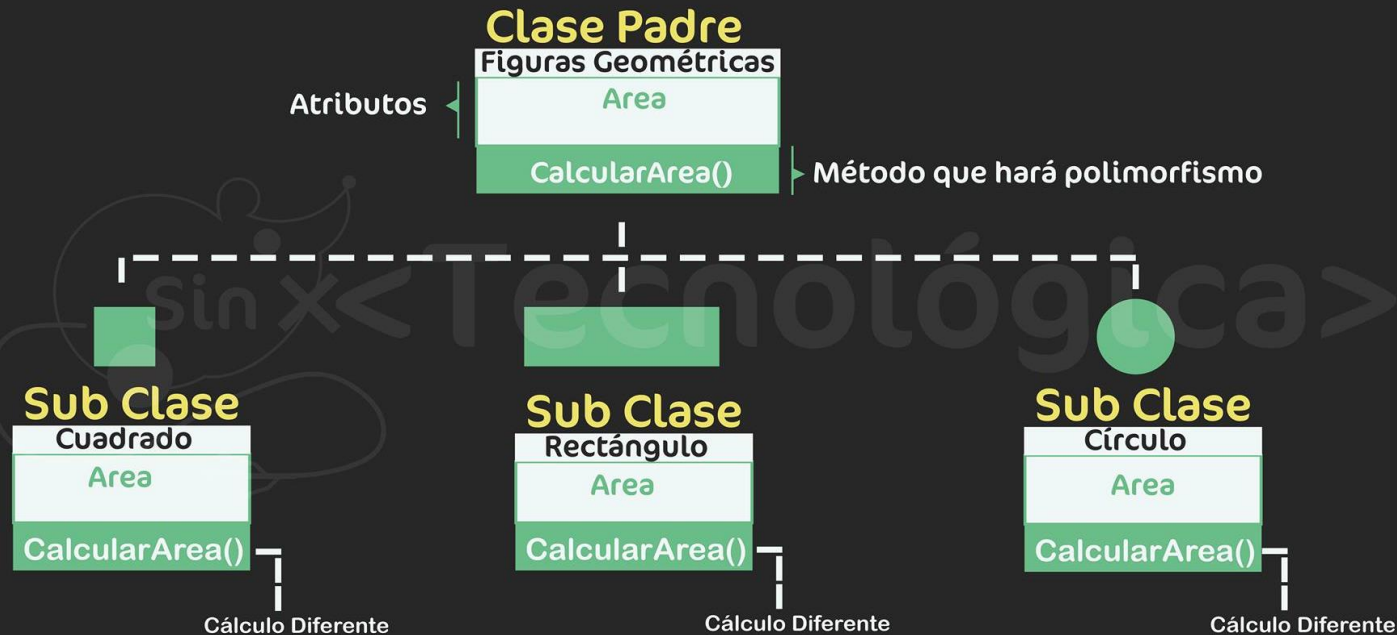
Debes ver este pilar de la Programación Orientado a objetos [POO] como una habilidad de modelado de datos, de esta forma tu estructura de datos estará mejor diseñada



Polimorfismo

¿Cómo funciona?

Describe múltiples y posibles estados de una única propiedad, en otras palabras: **consiste en conseguir que un objeto de una clase se comporte como un objeto de cualquiera de sus subclases**



Como en cada sub clase se calcula de forma diferente el área pero bajo el mismo nombre del método, cambiaremos el comportamiento del mismo con Polimorfismo

Clases

- La programación de objetos es el arte de definir y expandir clases.
- Una clase es un modelo de una parte muy específica de la realidad, que refleja las propiedades y actividades que se encuentran en el mundo real.
- Las clases definidas al principio son demasiado generales e imprecisas para cubrir el mayor número posible de casos reales.
- No hay obstáculo para definir nuevas subclases más precisas.
- Heredarán todo de su superclase, por lo que el trabajo que se utilizó para su creación no se desperdicia.

Clases

- La nueva clase puede agregar nuevas propiedades y nuevas actividades y, por lo tanto, puede ser más útil en aplicaciones específicas.
- Obviamente, se puede usar como una superclase para cualquier número de subclases recién creadas.
- El proceso no necesita tener un final. Puedes crear tantas clases como necesites.
- La clase que defina no tiene nada que ver con el objeto: la existencia de una clase no significa que ninguno de los objetos compatibles se creará automáticamente

Clases

- La clase en sí misma no puede crear un objeto: debe crearlo usted mismo y Python le permite hacerlo.
- Es hora de definir la clase más simple y crear un objeto.
- ejemplo:

```
class TheSimplestClass:  
  
    pass
```

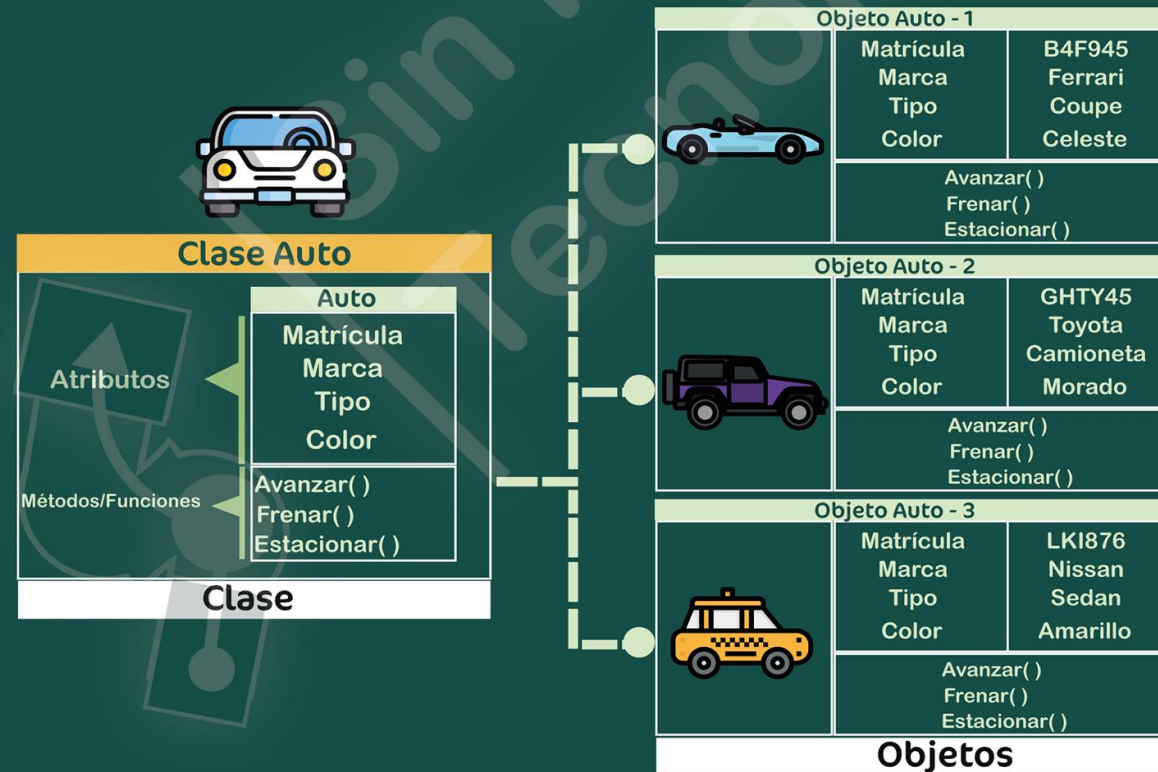
Clases

- La definición comienza con la palabra clave **class** .
- A la palabra clave le sigue un identificador que nombrará la clase (nota: no la confunda con el nombre del objeto; estas son dos cosas diferentes).
- A continuación, agrega dos puntos :, ya que las clases, como las funciones, forman su propio bloque anidado.
- El contenido dentro del bloque define todas las propiedades y actividades de la clase.
- La palabra clave **pass** llena la clase con nada. No contiene ningún método o propiedades.

Programación Orientada a Objetos

¿Qué es?

Es un paradigma de la programación, **organizado en clases de donde se crean objetos**. Está basado en la representación de objetos en la vida real.



Tu primer objeto

- La clase recién definida se convierte en una herramienta que puede crear nuevos objetos.
- La herramienta debe usarse explícitamente, bajo demanda.
- Imagine que desea crear un objeto (exactamente uno) de la

TheSimplestClassclass.

- Para hacer esto, debe asignar una variable para almacenar el objeto recién creado de esa clase y crear un objeto al mismo tiempo.
- Lo haces de la siguiente manera:

myFirstObject = TheSimplestClass()

Nota:

- El nombre de la clase intenta fingir que es una función.
- El objeto recién creado está equipado con todo lo que trae la clase; Como esta clase está completamente vacía, el objeto también está vacío.
- El acto de crear un objeto de la clase seleccionada también se denomina instanciación (ya que el objeto se convierte en una instancia de la clase).



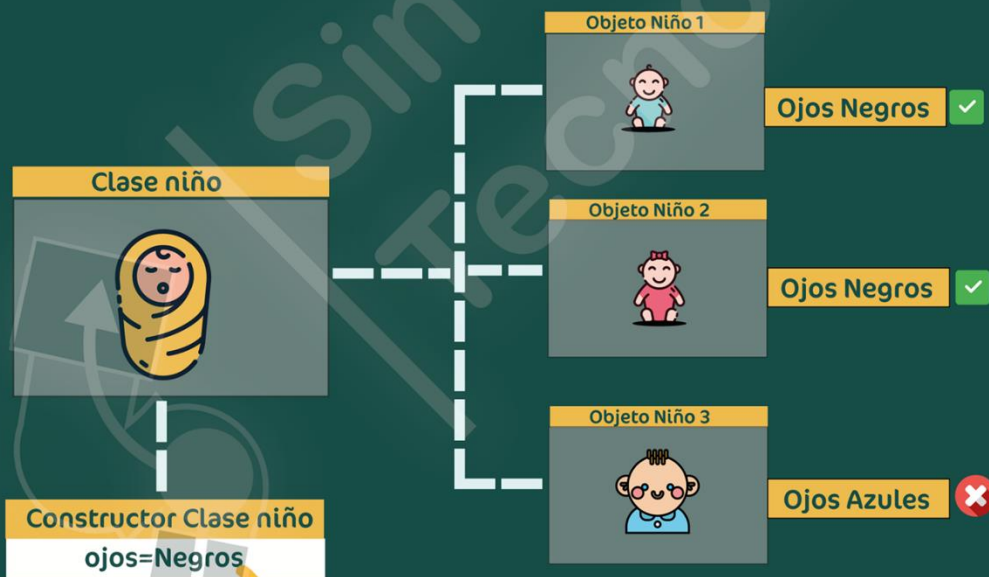
Constructor

¿Qué es?

Es un método especial de una clase que se ejecuta al momento de crear un objeto de la clase, **nos sirve principalmente para inicializar valores válidos en nuestro objeto al momento de crearlo.**

Ejemplo

Se entiende que todos los niños **por defecto nacen con color de ojos negros**



Se inicializaron los ojos de todos los niños al ser creados por defecto

Ejemplo

```
class Employee:
    'Common base class for all employees'
    empCount = 0
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1
    def displayCount(self):
        print "Total Employee %d" %
Employee.empCount
    def displayEmployee(self):
        print "Name : ", self.name, ", Salary:
", self.salary
```

Notas

- La variable **empCount** es una variable de clase cuyo valor se comparte entre todas las instancias de esta clase.
- Se puede acceder a este como **Employee.empCount** desde dentro de la clase o fuera de la clase.
- El primer método **__init__ ()** es un método especial, que se llama constructor de clases o método de inicialización que Python llama cuando crea una nueva instancia de esta clase.
- Usted declara otros métodos de clase como funciones normales con la excepción de que el primer argumento para cada método es **self**.
- Python agrega el argumento propio a la lista por usted; no necesita incluirlo cuando llame a los métodos.

Crear objetos de instancia

- Para crear instancias de una clase, llame a la clase usando el nombre de la clase y pase los argumentos que acepte su método `__init__` .

"This would create first object of Employee class"

```
emp1 = Employee("Zara", 2000)
```

"This would create second object of Employee class"

```
emp2 = Employee("Manni", 5000)
```

Acceso a atributos

- Accede a los atributos del objeto utilizando el operador de punto con objeto.
- Se accedería a la variable de clase utilizando el nombre de clase de la siguiente manera:

```
emp1.displayEmployee()
```

```
emp2.displayEmployee()
```

```
print ("Total Employee %d" %  
Employee.empCount)
```

- Cuando se ejecuta el código anterior, produce el siguiente resultado:

Name : Zara ,Salary: 2000

Name : Manni ,Salary: 5000

Total Employee 2

- Puede agregar, eliminar o modificar atributos de clases y objetos en cualquier momento:

```
emp1.age = 7 # Add an 'age' attribute.
```

```
emp1.age = 8 # Modify 'age' attribute.
```

```
del emp1.age # Delete 'age' attribute.
```

Funciones especiales

- En lugar de usar las declaraciones normales para acceder a los atributos, puede usar las siguientes funciones:
- El `getattr (obj, nombre [, predeterminado])` - para acceder al atributo del objeto.
- El `hasattr (obj, nombre)` : para verificar si un atributo existe o no.
- El `setattr (obj, nombre, valor)` - para establecer un atributo. Si el atributo no existe, entonces se crearía.
- El `delattr (obj, nombre)` : para eliminar un atributo.

Ejemplos

- `hasattr(emp1, 'age')` `#`
Returns true if 'age' attribute exists
- `getattr(emp1, 'age')` `#`
Returns value of 'age' attribute
- `setattr(emp1, 'age', 8)` `#`
Set attribute 'age' at 8
- `delattr(empl, 'age')` `#`
Delete attribute 'age'

Atributos de clase incorporados

- Cada clase de Python sigue los atributos incorporados y se puede acceder a ellos utilizando el operador de punto como cualquier otro atributo:

`__dict__` - Diccionario que contiene el espacio de nombres de la clase.

`__doc__` - Cadena de documentación de clase o ninguna, si no está definida.

`__name__` - Nombre de la clase.

`__module__` - Nombre del módulo en el que se define la clase. Este atributo es `"__main__"` en modo interactivo.

`__bases__` : una tupla posiblemente vacía que contiene las clases base, en el orden en que aparecen en la lista de clases base.

- Para la clase anterior intentemos acceder a todos estos atributos:

Ejemplo

```
class Employee:
    empCount = 0
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1
    def displayCount(self):
        print ("Total Employee %d" % Employee.empCount)
    def displayEmployee(self):
        print ("Name : ", self.name, " , Salary: ", self.salary)
print ("Employee.__doc__:", Employee.__doc__)
print ("Employee.__name__:", Employee.__name__)
print ("Employee.__module__:", Employee.__module__)
print ("Employee.__bases__:", Employee.__bases__)
print ("Employee.__dict__:", Employee.__dict__)
```

Destrucción de objetos (recolección de basura)

- Python elimina objetos innecesarios (tipos incorporados o instancias de clase) automáticamente para liberar espacio en la memoria.
- El proceso por el cual Python reclama periódicamente bloques de memoria que ya no están en uso se denomina recolección de basura.
- El recolector de basura de Python se ejecuta durante la ejecución del programa y se activa cuando el recuento de referencia de un objeto llega a cero.
- El recuento de referencia de un objeto cambia a medida que cambia el número de alias que lo señalan.
- El recuento de referencias de un objeto aumenta cuando se le asigna un nuevo nombre o se coloca en un contenedor (lista, tupla o diccionario).
- El recuento de referencias del objeto disminuye cuando se elimina con del , su referencia se reasigna o su referencia queda fuera de alcance.
- Cuando el recuento de referencia de un objeto llega a cero, Python lo recopila automáticamente.

Garbage Collections

¿Qué es?

Traducido como recolección de basura, es un mecanismo de lograr administrar memoria en un lenguaje de programación, dicho de otra forma: **Gestiona los espacios en memoria de los datos creados cuando programas como objetos, variables, etc.**

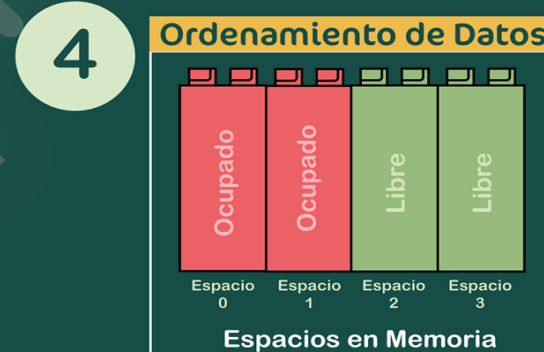
Funcionamiento del Garbage Collector



El Garbage Collector logra identificar los espacios usados y los espacios libres



Eliminación convencional



La compactación ordena los espacios en memoria, para su mejor rendimiento

Eliminación con compactación

Destrucción de objetos (recolección de basura)

a = 40 # Create object <40>

b = a # Increase ref. count of <40>

c = [b] # Increase ref. count of <40>

del a # Decrease ref. count of <40>

b = 100 # Decrease ref. count of <40>

c[0] = -1 # Decrease ref. count of <40>

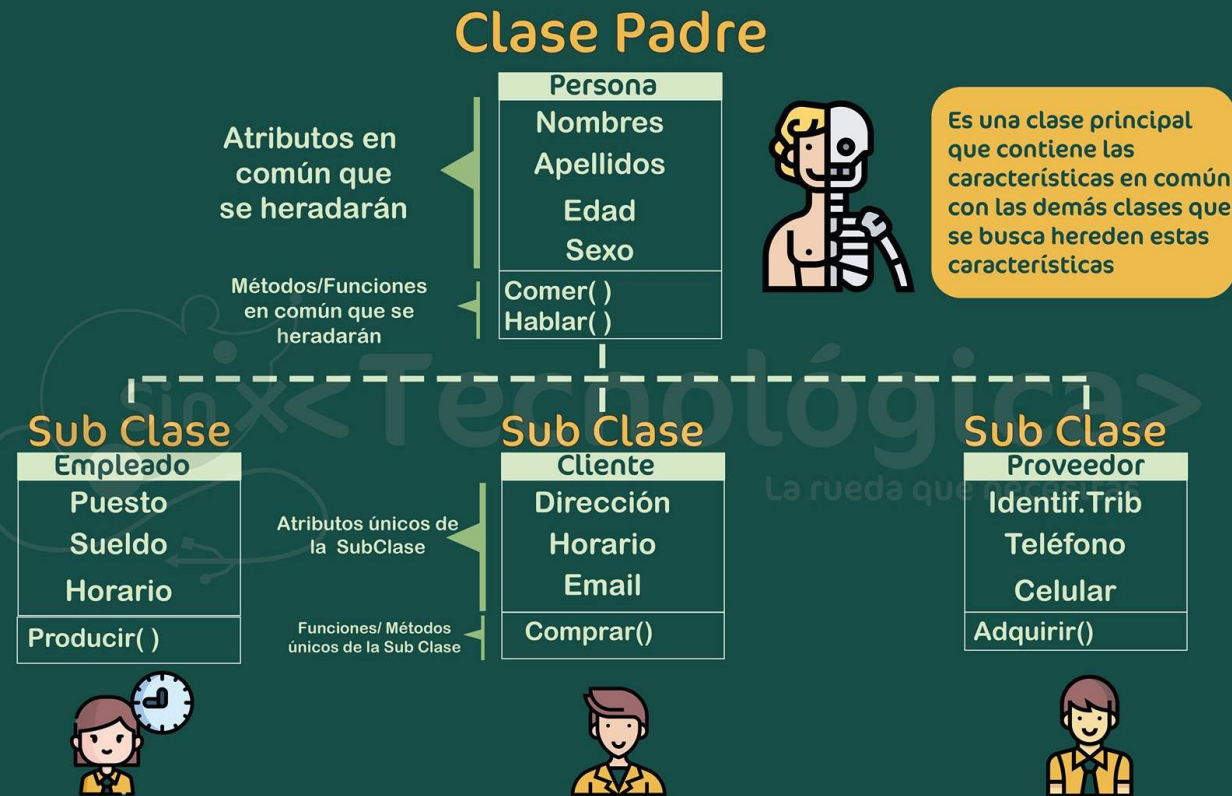
- Normalmente no se dará cuenta cuando el recolector de basura destruya una instancia huérfana y reclame su espacio.
- Pero una clase puede implementar el método especial `__del__()`, llamado destructor, que se invoca cuando la instancia está a punto de ser destruida.
- Este método puede usarse para limpiar cualquier recurso que no sea de memoria usado por una instancia.

Herencia de clase

- En lugar de comenzar desde cero, puede crear una clase derivándola de una clase preexistente enumerando la clase principal entre paréntesis después del nuevo nombre de clase.
- La clase secundaria hereda los atributos de su clase primaria, y puede usar esos atributos como si estuvieran definidos en la clase secundaria.
- Una clase secundaria también puede anular los miembros y métodos de datos del padre.

Herencia

¿Cómo funciona?



Las sub clases heredan las características de la clase padre, pero también pueden contener métodos/funciones y atributos únicos cada Sub Clase

Sintaxis

- Las clases derivadas se declaran como su clase principal; sin embargo, se proporciona una lista de clases base para heredar después del nombre de la clase:

```
class SubClassName
(ParentClass1[, ParentClass2, ...]):
    'Optional class documentation
    string'
    class_suite
```

- De manera similar, puede conducir una clase desde varias clases principales de la siguiente manera:

```
class A:    # define your class A
```

```
class B:    # define your class B
```

```
class C(A, B): # subclass of A and B
```

- Puede usar las funciones `issubclass ()` o `isinstance ()` para verificar las relaciones de dos clases e instancias.
 - La función booleana `issubclass (sub, sup)` devuelve verdadero si la subclase dada `sub` es de hecho una subclase de la superclase `sup` .
 - La función booleana `isinstance (obj, Class)` devuelve `true` si `obj` es una instancia de la clase `Class` o es una instancia de una subclase de `Class`

Métodos de anulación

- Siempre puede anular sus métodos de clase principal.
- Una razón para anular los métodos de los padres es porque es posible que desee una funcionalidad especial o diferente en su subclase.
- Ejemplo

```
class Parent:      # define parent class
    def myMethod(self):
        print ('Calling parent method')

class Child(Parent): # define child class
    def myMethod(self):
        print ('Calling child method')

c = Child()        # instance of child

c.myMethod()       # child calls overridden method
```

Métodos básicos de sobrecarga

- La siguiente tabla enumera algunas funciones genéricas que puede anular en sus propias clases:

No Señor.	Método, descripción y muestra de llamada
1	<code>__init__ (self [, args ...])</code> Constructor (con cualquier argumento opcional) Llamada de muestra: <code>obj = className (args)</code>
2	<code>__del__ (auto)</code> Destructor, elimina un objeto Llamada de muestra: <code>del obj</code>
3	<code>__repr__ (auto)</code> Representación de cadena evaluable Llamada de muestra: <code>repr (obj)</code>
4 4	<code>__str__ (auto)</code> Representación de cadena imprimible Llamada de muestra: <code>str (obj)</code>
5 5	<code>__cmp__ (auto, x)</code> Comparación de objetos Llamada de muestra: <code>cmp (obj, x)</code>

Operadores de sobrecarga

- Supongamos que ha creado una clase `vector` para representar vectores bidimensionales, ¿qué sucede cuando usa el operador “+” para agregarlos? Lo más probable es que Python nos entre algún mensaje.
- Sin embargo, podría definir el método `__add__` en su clase para realizar la suma de vectores y luego el operador “+” se comportaría según las expectativas:

ejemplo

```
class Vector:

    def __init__(self, a, b):

        self.a = a

        self.b = b

    def __str__(self):

        return ('Vector (%d, %d)' % (self.a, self.b))

    def __add__(self, other):

        return (Vector(self.a + other.a, self.b +
other.b))

v1 = Vector(2,10)

v2 = Vector(5,-2)

print (v1 + v2)
```

Ocultar datos (Encapsular)

- Los atributos de un objeto pueden o no ser visibles fuera de la definición de clase.
- Debe nombrar los atributos con un doble prefijo de subrayado, y esos atributos no serán directamente visibles para los extraños.

```
class JustCounter:
```

```
    __secretCount = 0
```

```
    def count(self):
```

```
        self.__secretCount += 1
```

```
        print (self.__secretCount)
```

```
counter = JustCounter()
```

```
counter.count()
```

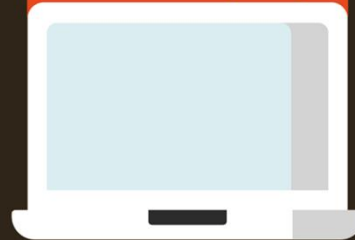
```
counter.count()
```

```
print (counter.__secretCount)
```

Encapsulamiento




¿Cómo funciona?

Clase Laptop



|

Atributos




Color: Blanco	
Marca: Apple	
Tamaño: 60cm	

Métodos

Públicos

Encender()	
Apagar()	
Reiniciar()	

Privados

Cambio de Memoria RAM()	
Cambio de HDD()	
Cambiar fuente de Energía()	

Encapsulados

La base del encapsulamiento se basa en que determinados métodos o atributos **no deben ser de acceso público por seguridad de manejo de datos**

Python protege a esos miembros cambiando internamente el nombre para incluir el nombre de la clase.

Puede acceder a atributos como `object._className__attrName`

Si modificamos el código con la siguiente instrucción el código corregirá el error.

```
print(counter._JustCounter__secretCount)
```



Generadores

- Un generador de Python es una pieza de código especializado capaz de producir una serie de valores y controlar el proceso de iteración .
- Es por eso que los generadores a menudo se llaman iteradores , y aunque algunos pueden encontrar una distinción muy sutil entre estos dos, los trataremos como uno.
- Puede que no te des cuenta, pero has encontrado generadores muchas, muchas veces antes. Eche un vistazo al fragmento muy simple:

```
for i in range(5):  
    print(i)
```

Generadores

¿Cuál es la diferencia?

- Una función devuelve un valor bien definido: puede ser el resultado de una evaluación más o menos compleja de, por ejemplo, un polinomio, y se invoca una vez, solo una vez.
- Un generador devuelve una serie de valores y, en general, se invoca (implícitamente) más de una vez.
- En el ejemplo, el generador `range()` se invoca seis veces, proporcionando cinco valores posteriores de cero a cuatro, y finalmente indicando que la serie está completa.
- El proceso anterior es completamente transparente.

Generadores

- El protocolo iterador es una forma en que un objeto debe comportarse para cumplir con las reglas impuestas por el contexto de las declaraciones **for** e **in** .
- Un objeto conforme al protocolo iterador se llama iterador .
- Un iterador debe proporcionar dos métodos:
- `__iter__()` que debería devolver el objeto en sí y que se invoca una vez (es necesario para que Python inicie con éxito la iteración)

Generadores

- `__next__()` que está destinado a devolver el siguiente valor (primero, segundo, etc.) de la serie deseada: las declaraciones `for/` lo invocarán `in` para pasar por la siguiente iteración; Si no hay más valores para proporcionar, el método debería generar la `StopIteration` excepción.

Ejemplo

```
class Fib:
    def __init__(self, nn):
        self.__n = nn
        self.__i = 0
        self.__p1 = self.__p2 = 1

    def __iter__(self):
        print("Fib iter")
        return self

    def __next__(self):
        self.__i += 1
        if self.__i > self.__n:
            raise StopIteration
        if self.__i in [1, 2]:
            return 1
        ret = self.__p1 + self.__p2
        self.__p1, self.__p2 = self.__p2, ret
        return ret

class Class:
    def __init__(self, n):
        self.__iter = Fib(n)

    def __iter__(self):
        print("Class iter")
        return self.__iter;

object = Class(8)

for i in object:
    print(i)
```

Acceso a archivos desde código Python

CEC-EPN

Juan Carlos Domínguez Ayala

02/2022



Introducción

- Uno de los problemas más comunes en el trabajo del desarrollador es procesar los datos almacenados en archivos, mientras que los archivos generalmente se almacenan físicamente utilizando dispositivos de almacenamiento: discos duros, ópticos, de red o de estado sólido.
- Es fácil imaginar un programa que clasifique 20 números, y es igualmente fácil imaginar que el usuario de este programa ingrese estos veinte números directamente desde el teclado.
- Es mucho más difícil imaginar la misma tarea cuando hay 20,000 números para ordenar, y no hay un solo usuario que pueda ingresar estos números sin cometer un error.

Introducción



En principio, cualquier problema de programación no simple se basa en el uso de archivos, ya sea que procese imágenes (almacenadas en archivos), multiplique matrices (almacenadas en archivos) o calcule salarios e impuestos (lectura de datos almacenados en archivos).

Open()

- La apertura de la secuencia se realiza mediante una función que se puede invocar de la siguiente manera:

stream = open(file, mode = 'r', encoding = None)



- El nombre de la función (open) habla por sí mismo; si la apertura es exitosa, la función devuelve un objeto de flujo; de lo contrario, se genera una excepción (por ejemplo, FileNotFoundError si el archivo que va a leer no existe)

```
stream = open(file, mode = 'r',  
encoding = None)
```

1. el primer parámetro de la función (**file**) especifica el nombre del archivo que se asociará a la secuencia.
2. el segundo parámetro (**mode**) especifica el modo abierto utilizado para la secuencia; es una cadena llena de una secuencia de caracteres, y cada uno de ellos tiene su propio significado especial
3. el tercer parámetro (**encoding**) especifica el tipo de codificación (por ejemplo, UTF-8 cuando se trabaja con archivos de texto)

NOTAS

- primera operación realizada en la transmisión.
- El modo y los argumentos de codificación pueden omitirse; sus valores predeterminados se suponen entonces.
- El modo de apertura predeterminado es leer en modo de texto, mientras que la codificación predeterminada depende de la plataforma utilizada.

Modos de abrir archivo texto

1. **r modo abierto: leer**
 - la secuencia se abrirá en modo de lectura ;
 - el archivo asociado con la secuencia debe existir y debe ser legible; de lo contrario, la función open() genera una excepción.
2. **w modo abierto: escribir**
 - la secuencia se abrirá en modo de escritura ;
 - el archivo asociado con la secuencia no necesita existir ; si no existe, se creará; si existe, se truncará a la longitud de cero (borrado); Si la creación no es posible (por ejemplo, debido a los permisos del sistema), la función open() genera una excepción.
3. **a modo abierto: agregar**
 - la secuencia se abrirá en modo agregar ;
 - el archivo asociado con la secuencia no necesita existir ; si no existe, se creará; si existe, el cabezal de grabación virtual se establecerá al final del archivo (el contenido anterior del archivo permanece intacto).

Modos de abrir archivo texto

1. **r+ modo abierto: leer y actualizar**
 - la transmisión se abrirá en modo de lectura y actualización ;
 - el archivo asociado con la secuencia debe existir y debe poder escribirse ; de lo contrario, la función open() genera una excepción;
 - Se permiten las operaciones de lectura y escritura para la secuencia.
2. **w+ modo abierto: escribir y actualizar**
 - la transmisión se abrirá en modo de escritura y actualización ;
 - el archivo asociado con la secuencia no necesita existir ; si no existe, se creará; el contenido anterior del archivo permanece intacto;
 - Se permiten las operaciones de lectura y escritura para la secuencia.

Tabla de resumen

Modo de texto	Modo binario	Descripción
<code>rt</code>	<code>rb</code>	leer
<code>wt</code>	<code>wb</code>	escribir
<code>at</code>	<code>ab</code>	adjuntar
<code>r+t</code>	<code>r+b</code>	leer y actualizar
<code>w+t</code>	<code>w+b</code>	escribir y actualizar

close()

- La última operación realizada en el manejo de archivos es **close()** (esto no incluye los stdin, stdout y stderr corrientes que no lo requieran).
- Esa acción es realizada por un método invocado desde dentro de objeto de flujo abierto: ejemplo **file.close()**.
- el nombre de la función definitivamente se autocomenta (close())
- la función no espera exactamente argumentos; la secuencia no necesita ser abierta
- la función no devuelve nada pero genera una excepción IOError en caso de error;
- La mayoría de los desarrolladores creen que la función close() siempre tiene éxito y, por lo tanto, no hay necesidad de verificar si ha realizado su tarea correctamente.

close()

- Esta creencia está solo parcialmente justificada.
- Si el flujo se abrió para escritura y luego se realizó una serie de operaciones de escritura, puede suceder que los datos enviados al flujo aún no se hayan transferido al dispositivo físico (debido al mecanismo llamado almacenamiento en caché o almacenamiento en búfer).
- Dado que el cierre de la corriente obliga a los búferes a descargarlos, puede ser que el proceso de limpieza fallen y, por lo tanto, close() también falle.
- Ya hemos mencionado fallas causadas por funciones que funcionan con flujos, pero no mencionamos una palabra sobre cómo podemos identificar exactamente la causa de la falla que se deben contener con los procesos de excepciones.

Trabajos con archivos



