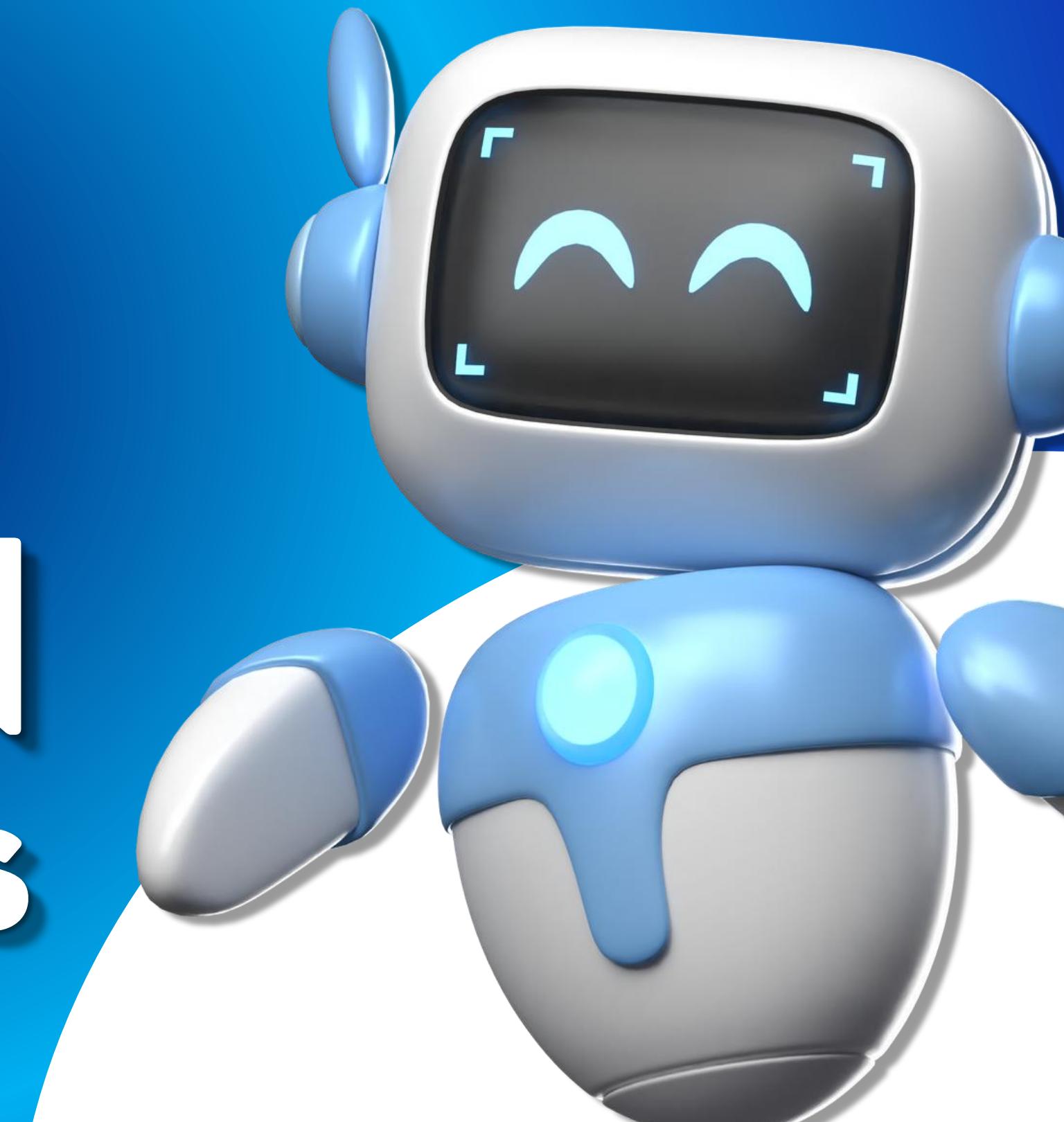




IA PYTHON PARA PRINCIPIANTES

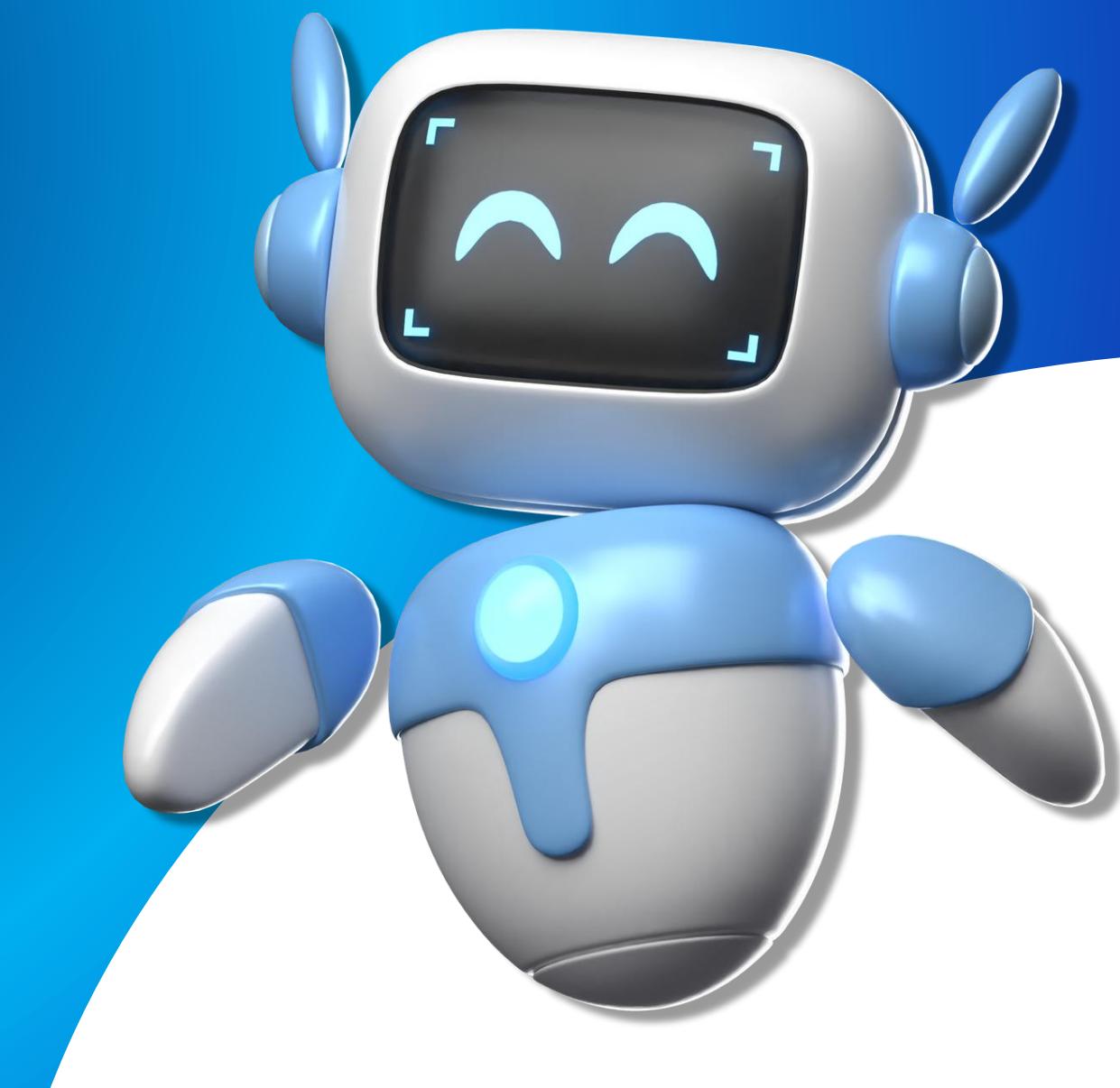




IA PYTHON PARA PRINCIPIANTES

CLASE 11

Programación Orientada a
Objetos



CONTENIDO



01

Usando GIT

02

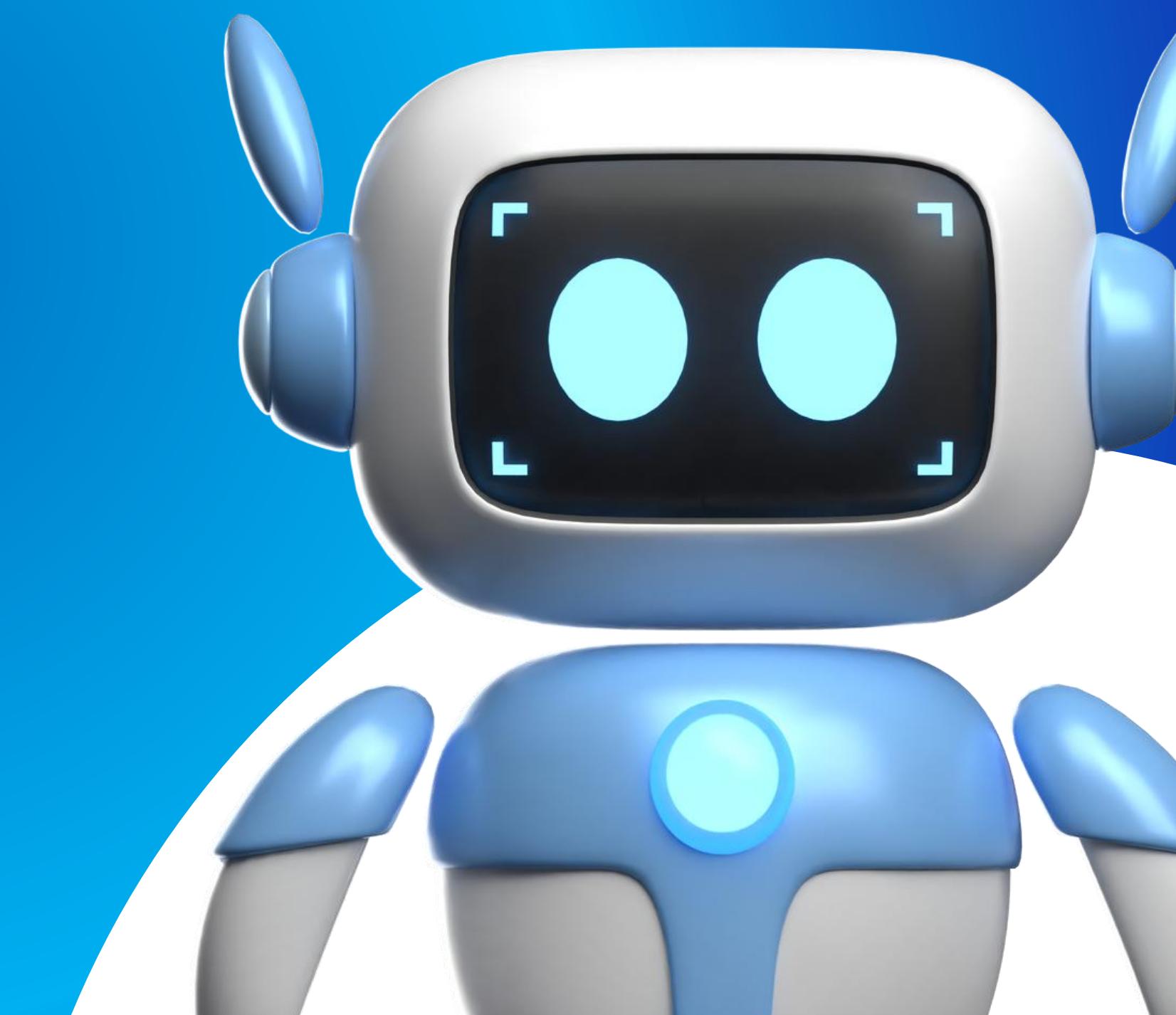
¿Que es POO?

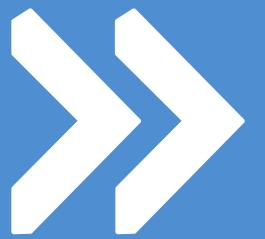
03

Terminología y Clases en Python

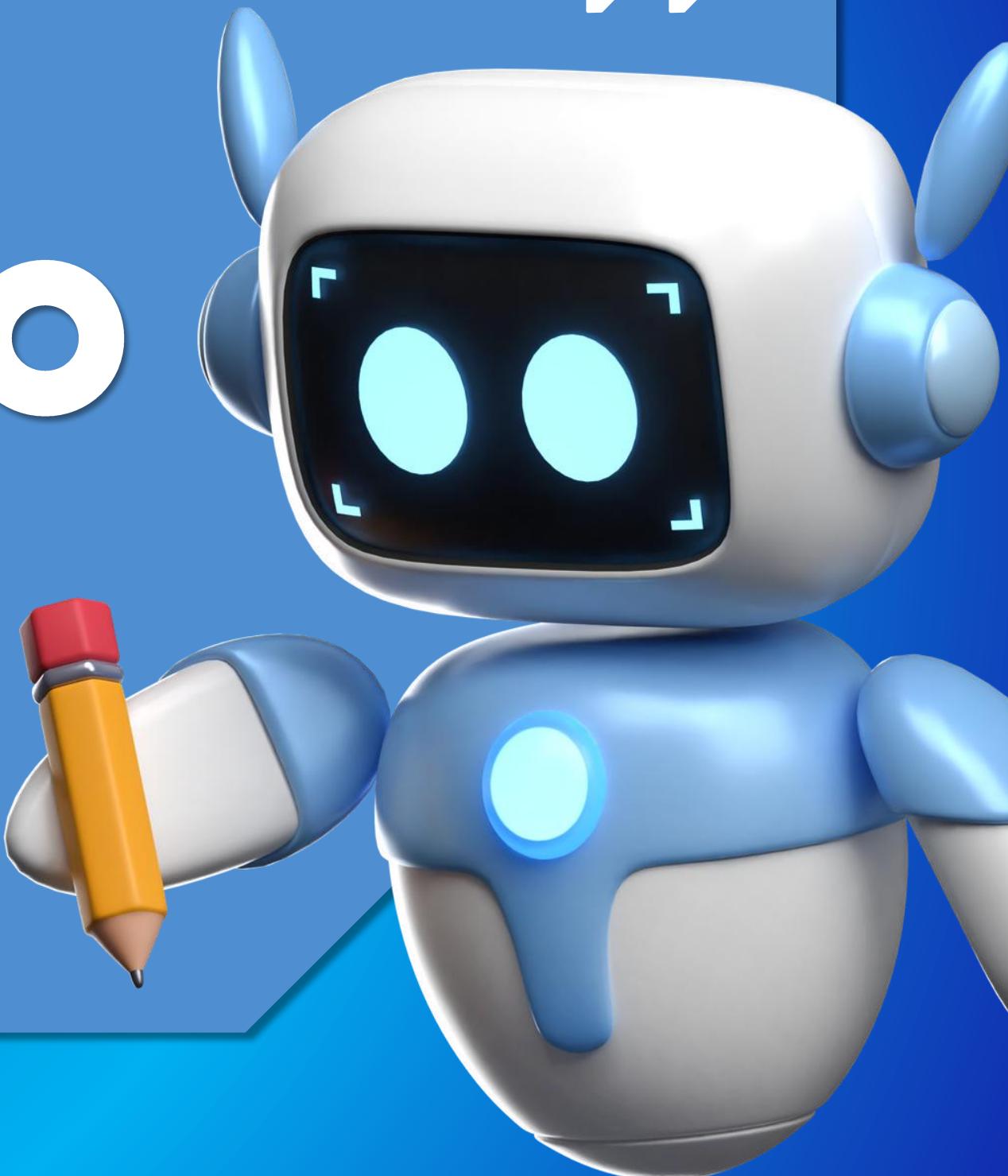
04

Ciclo de vida de un Objeto





GIT INTEGRANDO LO APRENDIDO



COMANDOS BASICOS GIT

Git es una herramienta fundamental para cualquier desarrollador, ya que permite controlar versiones de código, colaborar con otros y mantener un historial de cambios.

PREGUNTALE A UN LLM:

- **¿CÓMO INSTALAR Y COMO HACER LA CONFIGURACION INICIAL?**
- **¿CÓMO INICIALIZAR UN REPOSITORIO Y COMO CLONAR?**
- **¿CUÁLES SON LOS COMANDOS BASICOS?**
- **¿QUÉ SON LAS RAMAS Y COMO SE UTILIZAN?**
- **GIT VS GITHUB.**



Herramienta de control de versiones distribuida

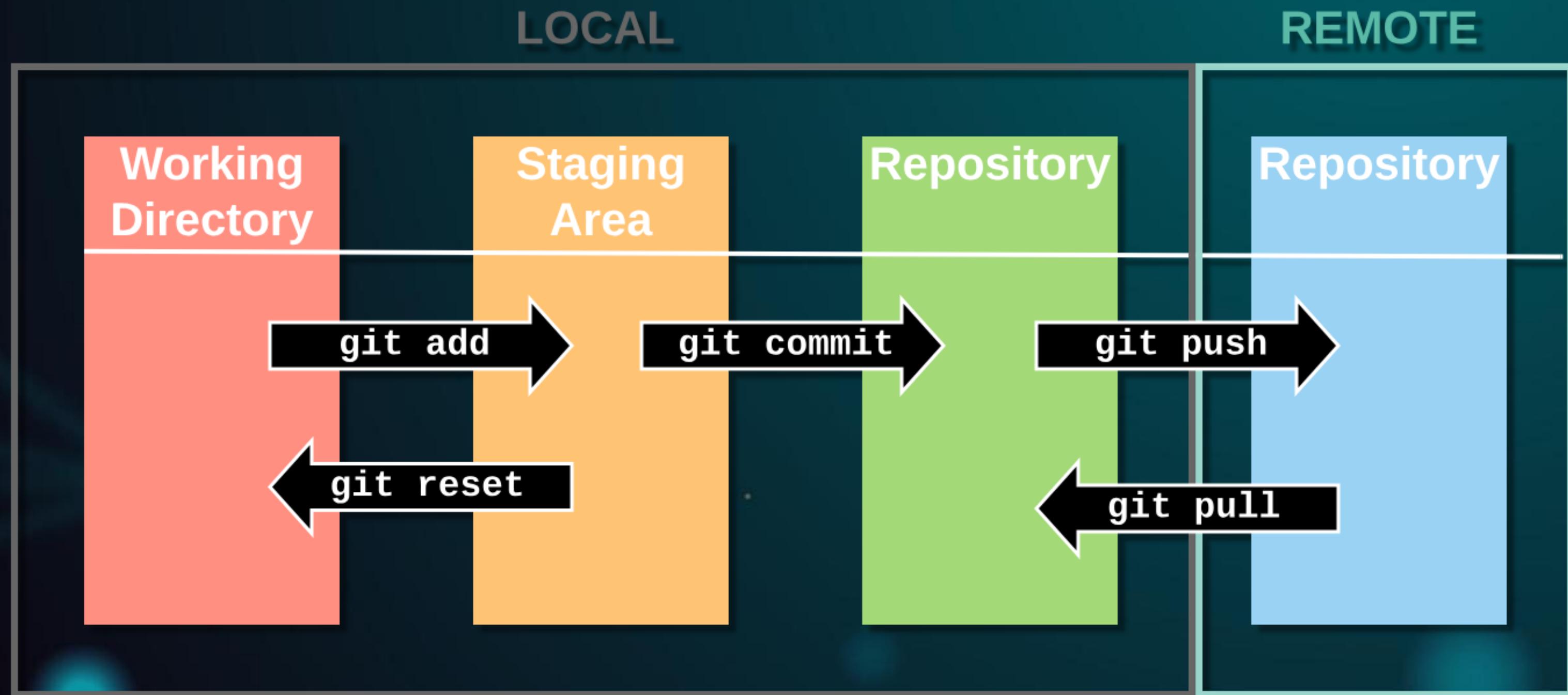
Herramienta de código abierto que los desarrolladores instalan localmente para gestionar el código fuente



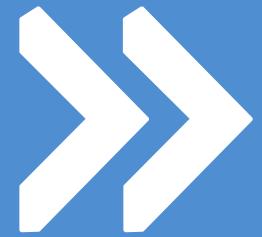
Plataforma basada en la nube

Servicio en línea al que los desarrolladores que utilizan Git pueden conectarse y cargar o descargar recursos

FLUJO DE TRABAJO EN GIT



Para instalar Git: <https://git-scm.com/downloads>



PROGRAMACION ORIENTADA A OBJETOS



¿QUE ES ORIENTADO A OBJETO?

La programación orientada a objetos (POO) es un paradigma de programación que se basa en el concepto de “objetos”.

En lugar de tener un flujo de control lineal, en POO, el programa se compone de **objetos individuales** que poseen ciertas **capacidades** y realizan tareas específicas. Estos **objetos** “cooperan” entre sí para completar el programa.

Cada objeto es como una “isla” que tiene su propio estado y comportamiento, y puede interactuar con otros objetos.

Un “**objeto**” puede ser la representación de una entidad de nuestra vida real. Para crear un **objeto** necesitamos una “**clase**” o plantilla. Una **clase** representa las **características** en común de nuestros “**objetos**”.

Python es un lenguaje de programación orientado a objetos.

```
0  response = re
1  # checking re
2  if response.s
3  print(f"S
4  else:
5  | print(f"S
6
7  # using Beau
8  soup = Beau
9
10 # finding Pos
11 images = sou
12
13 # downloadin
14 count = 0
```

OBJETOS

The laptop screen displays the following content:

5. Data Structures

This chapter describes some things you've learned about already in more detail, and adds some new things as well.

5.1. More on Lists

The `list` data type has some more methods. Here are all of the methods of list **objects**:

list.append(x)
Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

list.extend(L)
Extend the list by appending all the items in the given list. Equivalent to `a[len(a):] = L`.

list.insert(i, x)
Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

list.remove(x)
Remove the first item from the list whose value is `x`. It is an error if there is no such item.

list.pop([i])
Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the `i` in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

OBJETOS

12.6. `sqlite3` — DB-API 2.0 interface for SQLite databases

[Source code: Lib/sqlite3/](#)

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The `sqlite3` module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by [PEP 249](#).

To use the module, you must first create a `Connection` object that represents the database. Here the data will be stored in the `example.db` file:

```
import sqlite3
conn = sqlite3.connect('example.db')
```

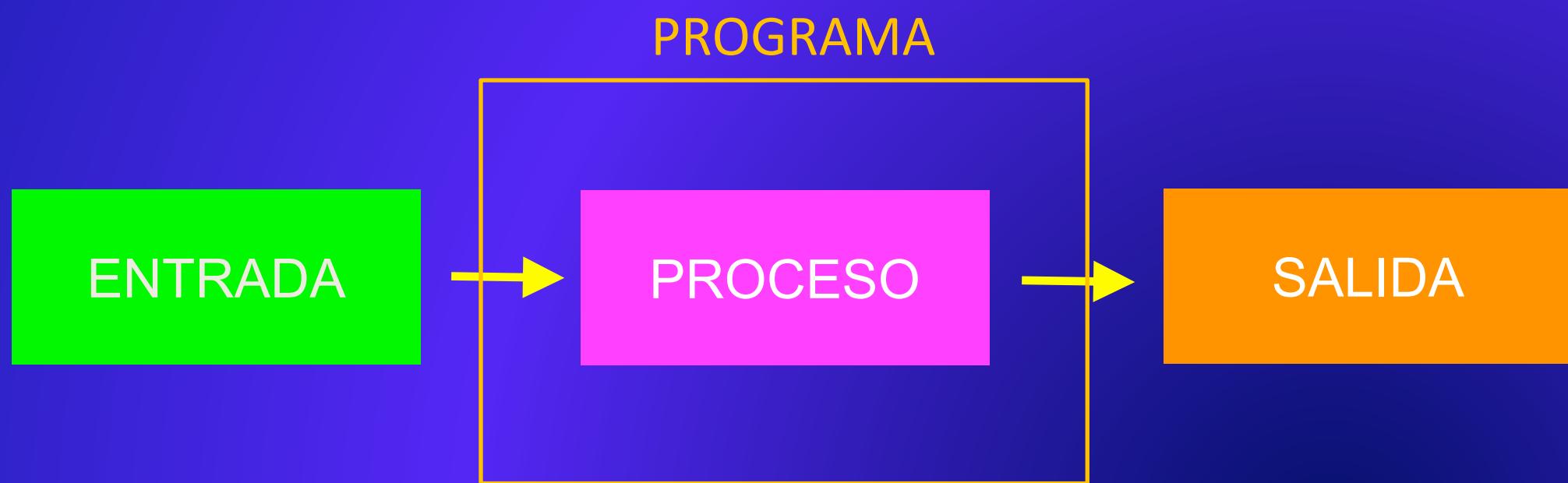
You can also supply the special name `:memory:` to create a database in RAM.

Once you have a `Connection`, you can create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
c = conn.cursor()

# Create table
c.execute('''CREATE TABLE stocks
            (date text, trans text, symbol text, qty real, price real)''')
```

OBJETOS

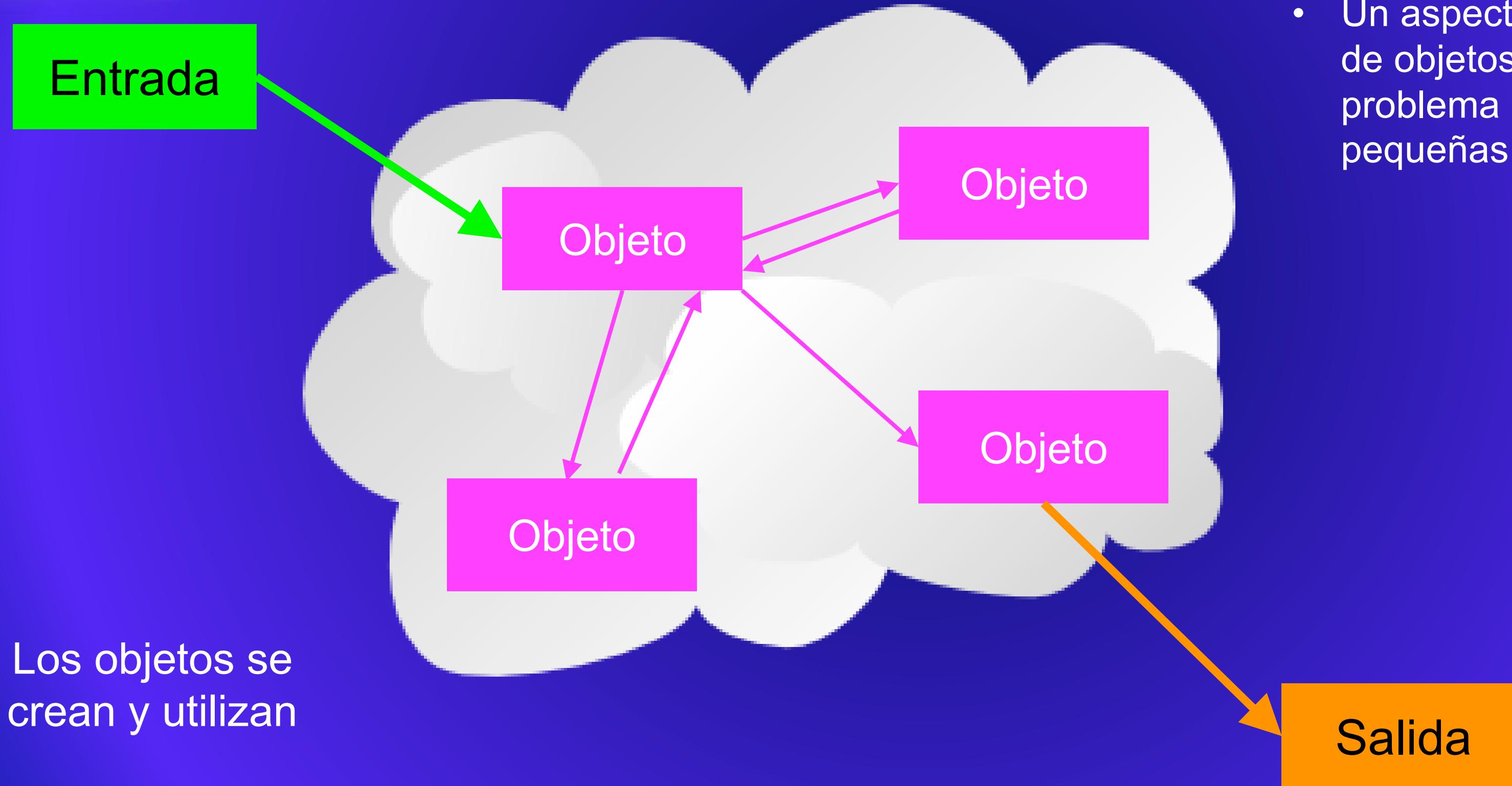


En la programación secuencial o funcional tenemos un control lineal del proceso. Manejamos estructuras de control para dirigir el flujo. O bien encapsulábamos en funciones cierta parte del código que utilizaríamos repetidamente.

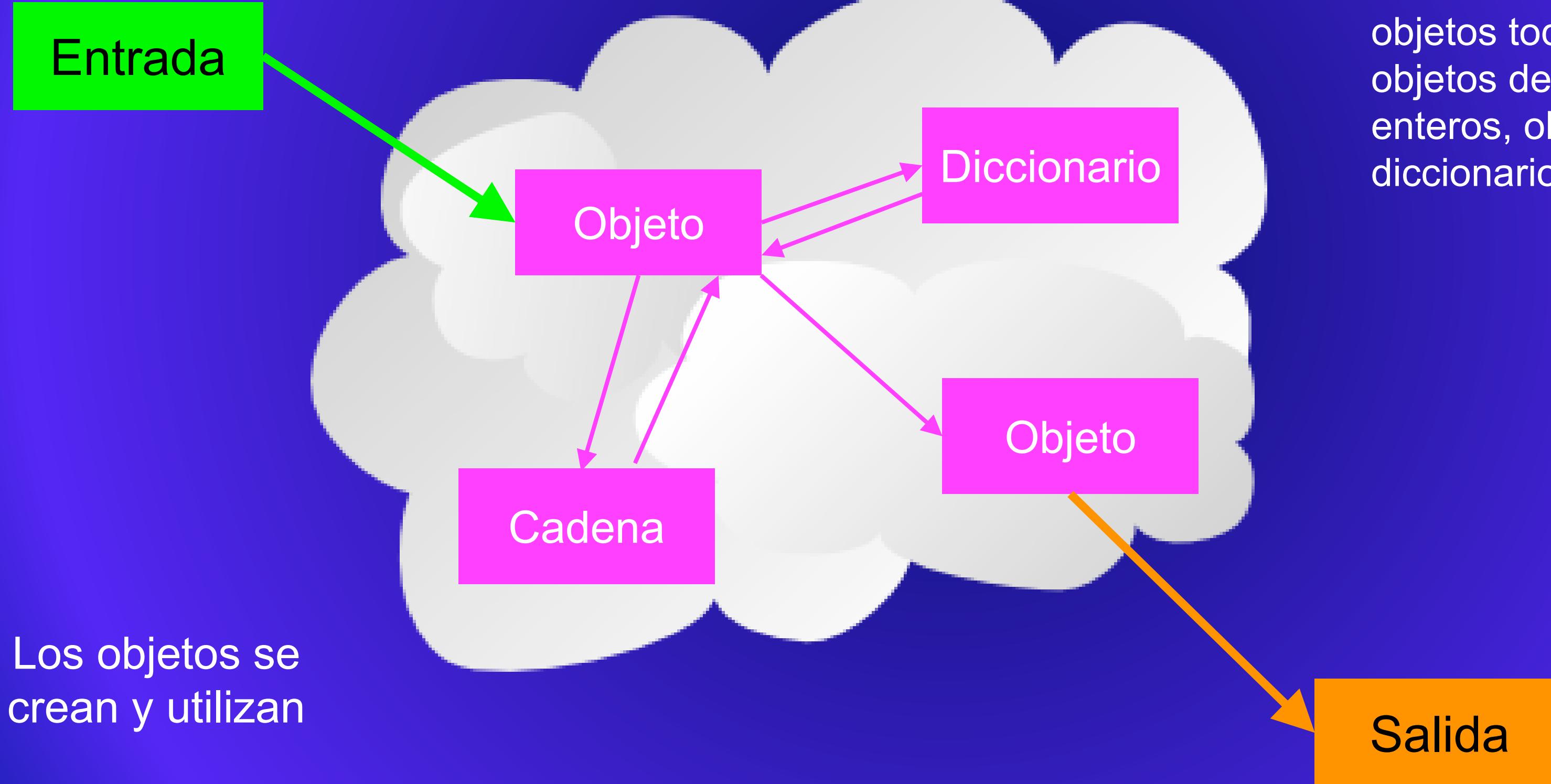
- Un objeto es un poco de código y datos autónomos



DIVIDE Y VENCERÁS



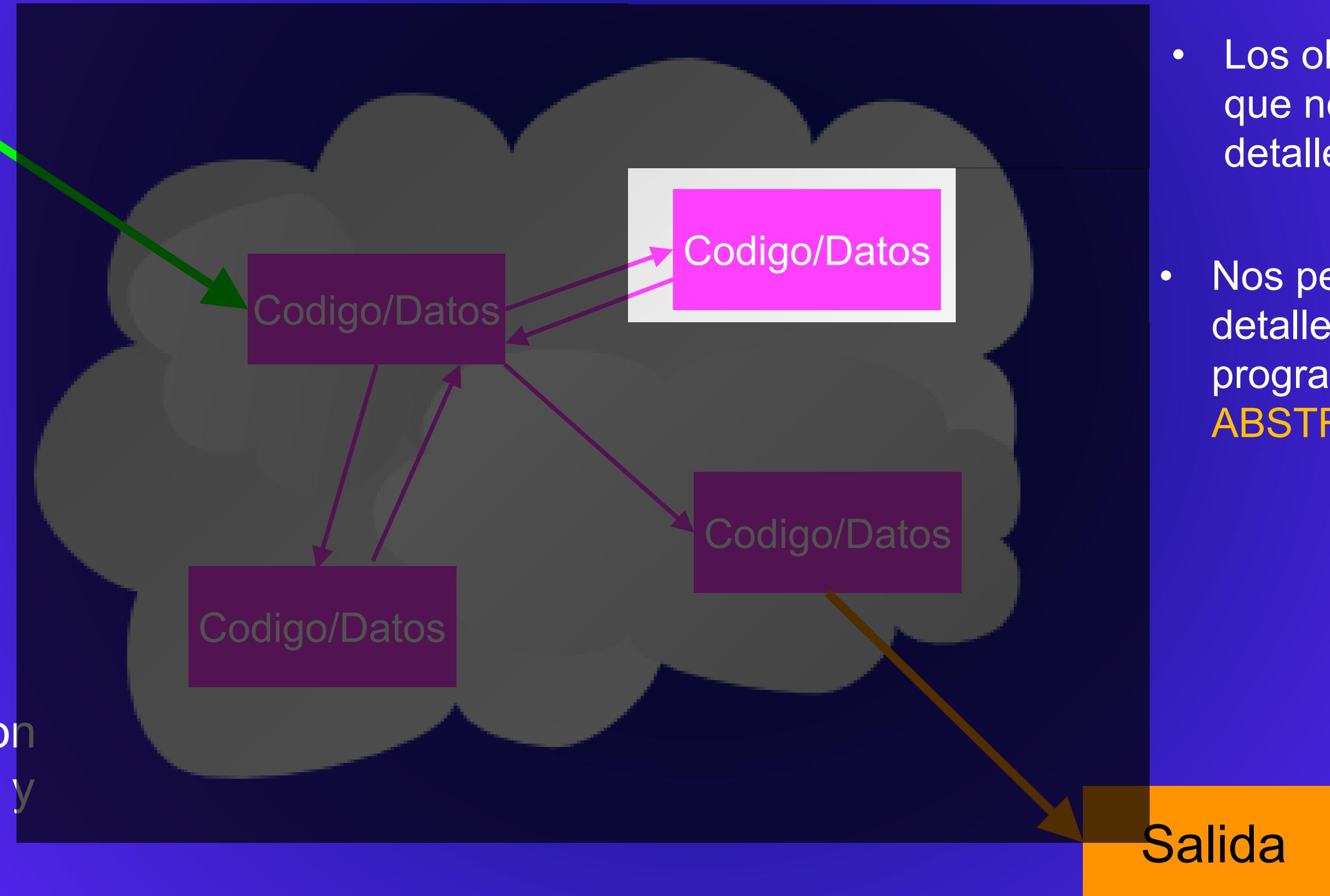
DIVIDE Y VENCERÁS



- Hemos estado usando objetos todo el tiempo: objetos de cadena, objetos enteros, objetos de diccionario, objetos de lista...

DIVIDE Y VENCERÁS

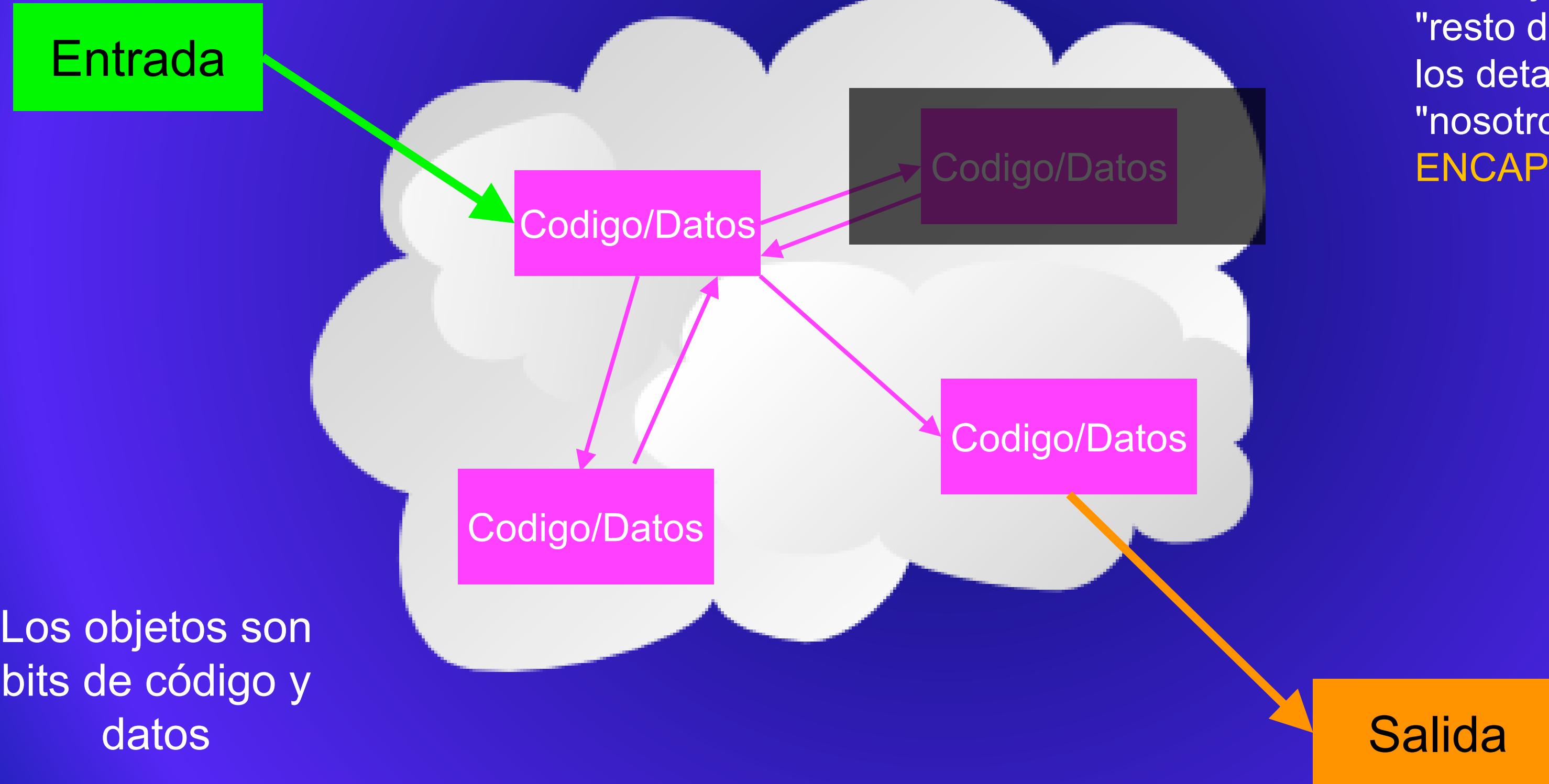
Entrada



Los objetos son
bits de código y
datos

- Los objetos tienen límites que nos permiten ignorar detalles innecesarios.
- Nos permiten ignorar el detalle del "resto del programa".
ABSTRACCIÓN

DIVIDE Y VENCERÁS



- Los objetos permiten que el "resto del programa" ignore los detalles sobre "nosotros".
ENCAPSULAMIENTO

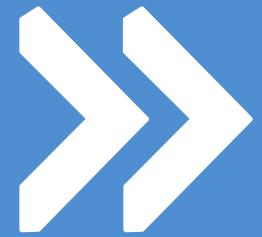
DEFINICIONES

CLASE: Una clase es una **plantilla** o un **plano** que define un conjunto de atributos y métodos que caracterizan a cualquier objeto de esa clase.

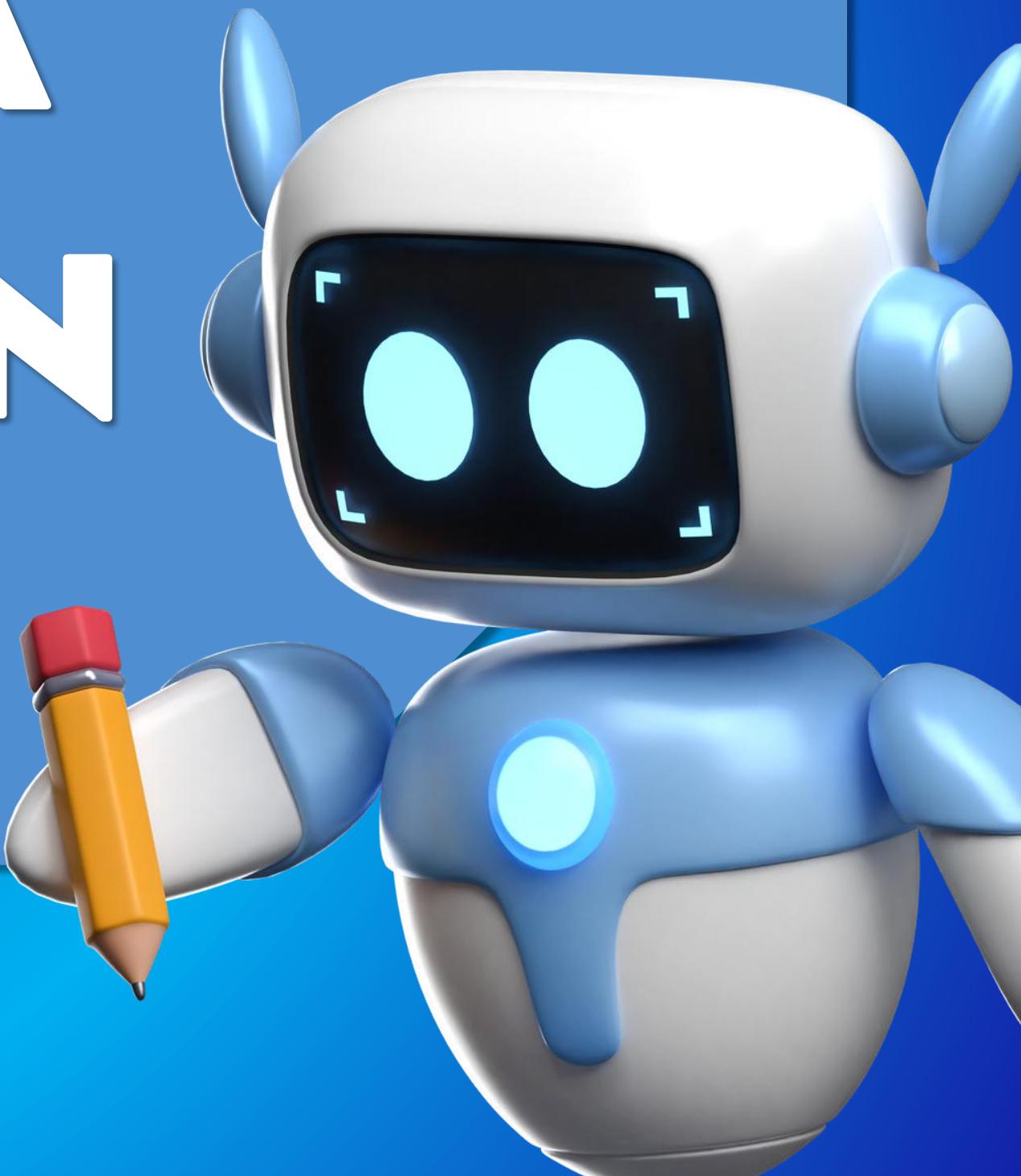
METODOS O MENSAJES: Los métodos son funciones definidas dentro de una clase y se utilizan para realizar acciones específicas en los objetos creados a partir de esa clase.

CAMPO O ATRIBUTO: Los atributos son variables que almacenan información sobre el estado de un objeto. pueden ser de dos tipos: atributos de instancia (pertenecen a cada instancia de la clase) y atributos de clase (pertenecen a la clase en sí y se comparten entre todas las instancias de la clase)

OBJETO: Es una instancia particular de una CLASE. En Python todo es un objeto. Los objetos son la base de todo, cada entidad, desde números simples hasta estructuras de datos complejas, son representadas como objetos que tienen datos y comportamientos asociados.



TERMINOLOGÍA DE OBJETOS EN MAS DETALLE



TERMINOLOGÍA: “CLASE”

Define las características abstractas de una cosa (objeto), incluidas las características de la cosa (sus atributos, **campos** o **propiedades**) y los comportamientos de la cosa (las cosas que puede hacer, o **métodos**, operaciones o características). Se podría decir que una **clase** es un **plano** o fábrica que describe la naturaleza de algo. Por ejemplo, la **clase Perro** consistiría en rasgos compartidos por todos los perros, como la raza y el color del pelaje (características), y la capacidad de ladrar y sentarse (comportamientos).

D
A
T
O
S

C
O
D
I
G
O

Clase PERRO

ATRIBUTOS:

Raza
Color
Edad

METODOS:

Ladrar()
Sentarse()
Correr()

TERMINOLOGÍA: “INSTANCIA”

Uno puede tener una **instancia** de una clase o un objeto particular. La **instancia** es el objeto real creado en tiempo de ejecución. En la jerga del programador, el objeto Lassie es una **instancia** de la clase PERRO. El conjunto de valores de los atributos de un **objeto** determinado se denomina **estado**. El **objeto** consta del estado y el comportamiento definido en la clase del objeto.

Clase PERRO

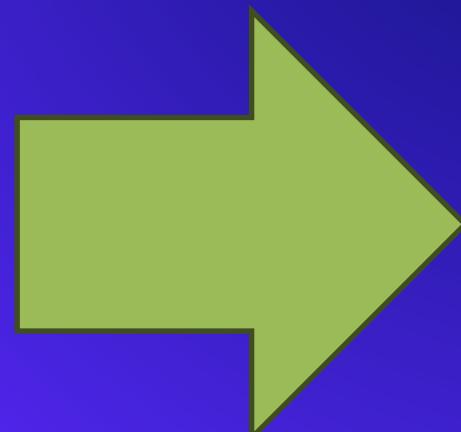
ATRIBUTOS:

Raza
Color
Edad

METODOS:

Ladrar()
Sentarse()
Correr()

Objeto e Instancia a
menudo se usan
indistintamente.



lassie

ATRIBUTOS:

Raza Collie
Color marrón
Edad 2

METODOS:

Ladrar()
Sentarse()
Correr()

tony

ATRIBUTOS:

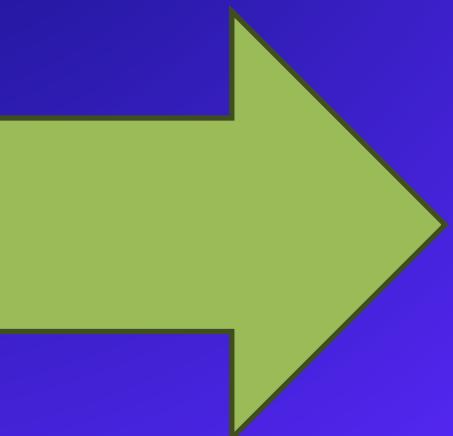
Raza labrador
Color blanco
Edad 4

METODOS:

Ladrar()
Sentarse()
Correr()

TERMINOLOGÍA: “MÉTODO”

Las habilidades de un objeto. En el lenguaje, los **métodos** son verbos. Lassie, siendo un perro, tiene la capacidad de ladrar. Así que ladrar() es uno de los métodos de Lassie. También puede tener otros **métodos**, por ejemplo, sentarse () o comer () o caminar () o correr(). Dentro del programa, el uso de un **método** generalmente afecta solo a un objeto en particular; todos los perros pueden ladrar, pero solo necesitas un perro en particular para ladrar



Método y Mensaje a menudo se usan indistintamente.

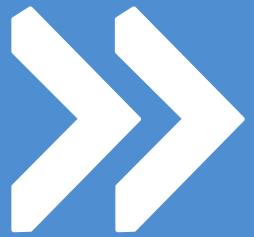
lassie

ATRIBUTOS:

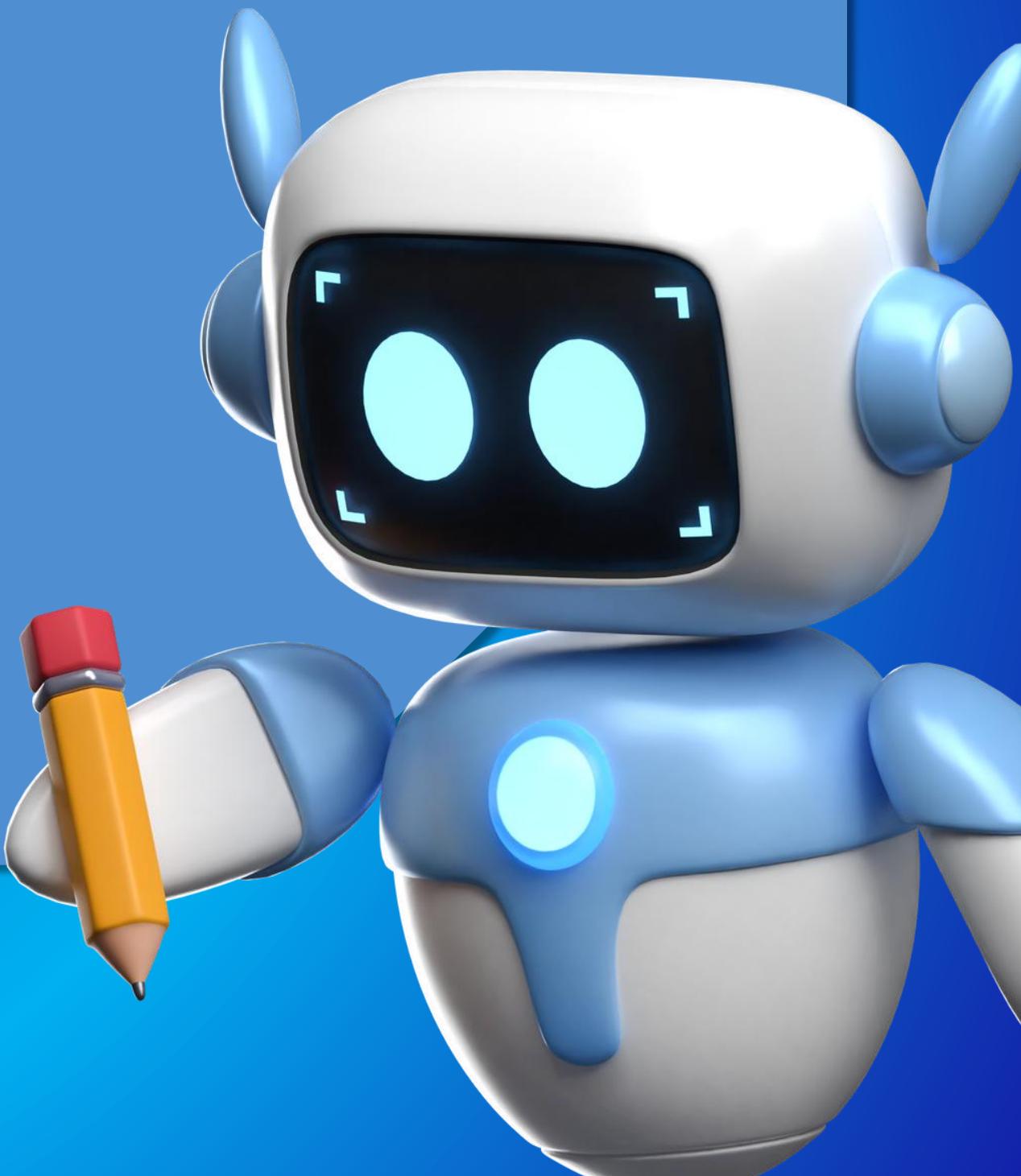
Raza	Collie
Color	marrón
Edad	2

MÉTODOS:

- Ladrar()
- Sentarse()
- Correr()



CLASES EN PYTHON



ALGUNOS OBJETOS DE PYTHON

```
>>> x = 'abc'          >>> dir(x)
>>> type(x)           [ ... 'capitalize', 'casefold', 'center', 'count',
<class 'str'>          'encode', 'endswith', 'expandtabs', 'find', 'format',
                         ... 'lower', 'lstrip', 'maketrans', 'partition',
>>> type(2.5)          'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
<class 'float'>         'rsplit', 'rstrip', 'split', 'splitlines',
                           'startswith', 'strip', 'swapcase', 'title',
>>> type(2)             'translate', 'upper', 'zfill']
<class 'int'>

>>> y = list()          >>> dir(y)
>>> type(y)              [... 'append', 'clear', 'copy', 'count', 'extend',
<class 'list'>            'index', 'insert', 'pop', 'remove', 'reverse',
                           'sort']

>>> z = dict()          >>> dir(z)
>>> type(z)              [..., 'clear', 'copy', 'fromkeys', 'get', 'items',
<class 'dict'>            'keys', 'pop', 'popitem', 'setdefault', 'update',
                           'values']
```

CLASE DE EJEMPLO

class es una
palabra reservada

Cada objeto
PartyAnimal tiene un
poco de código

Indique al objeto an
que ejecute el código
party() dentro de él

```
class PartyAnimal:  
    x = 0  
  
    def party(self) :  
        self.x = self.x + 1  
        print("Hasta ahora",self.x)  
  
an = PartyAnimal()  
  
an.party()  
an.party()  
an.party()
```

Esta es la plantilla
para crear objetos
PartyAnimal

Cada objeto
PartyAnimal tiene
un poco de datos

Construir un objeto
PartyAnimal y
almacenarlo en an

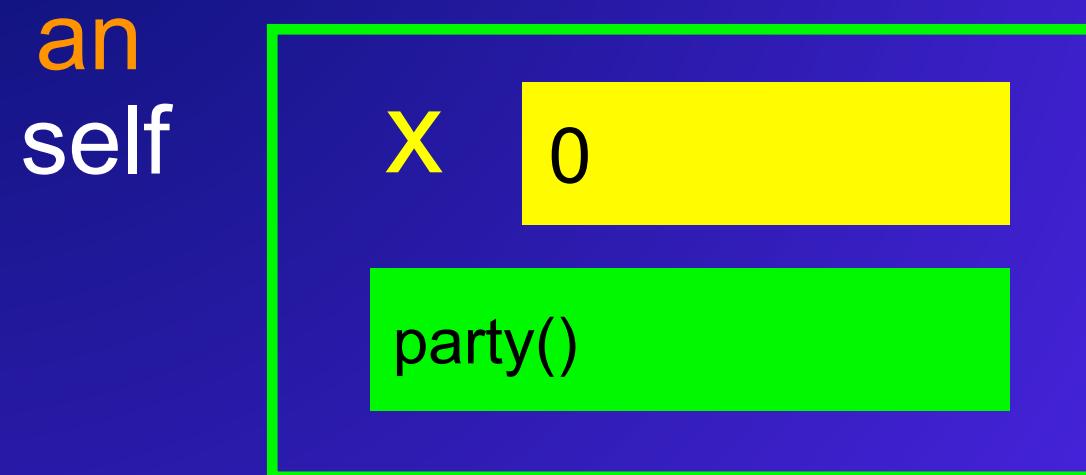
PartyAnimal.party(an)

CLASE DE EJEMPLO

```
class PartyAnimal:  
    x = 0  
  
    def party(self):  
        self.x = self.x + 1  
        print("Hasta ahora", self.x)  
  
an = PartyAnimal()  
  
an.party()  
an.party()  
an.party()
```

\$ python party.py

Hasta ahora 1
Hasta ahora 2
Hasta ahora 3



Una forma Nerd de encontrar capacidades

- El comando `dir()` enumera las capacidades
- Ignora los que tienen guiones bajos: estos son utilizados por el propio Python
- El resto son operaciones reales que el objeto puede realizar
- Es como `type()` - nos dice algo *sobre* una variable

```
>>> y = list()
>>> type(y)
<class 'list'>
>>> dir(y)
['__add__', '__class__', '__contains__',
 '__delattr__', '__delitem__',
 '__delslice__', '__doc__', ...
 '__getitem__', '__setslice__', '__str__',
 'append', 'clear', 'copy', 'count',
 'extend', 'index', 'insert', 'pop',
 'remove', 'reverse', 'sort']
>>>
```



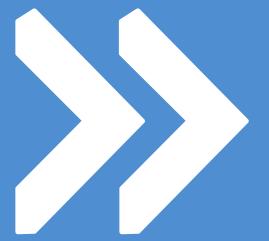
Una forma Nerd de encontrar capacidades

```
class PartyAnimal:  
    x = 0  
  
    def party(self) :  
        self.x = self.x + 1  
        print("Hasta  
ahora",self.x)  
  
an = PartyAnimal()  
  
print("Type", type(an))  
print("Dir ", dir(an))
```

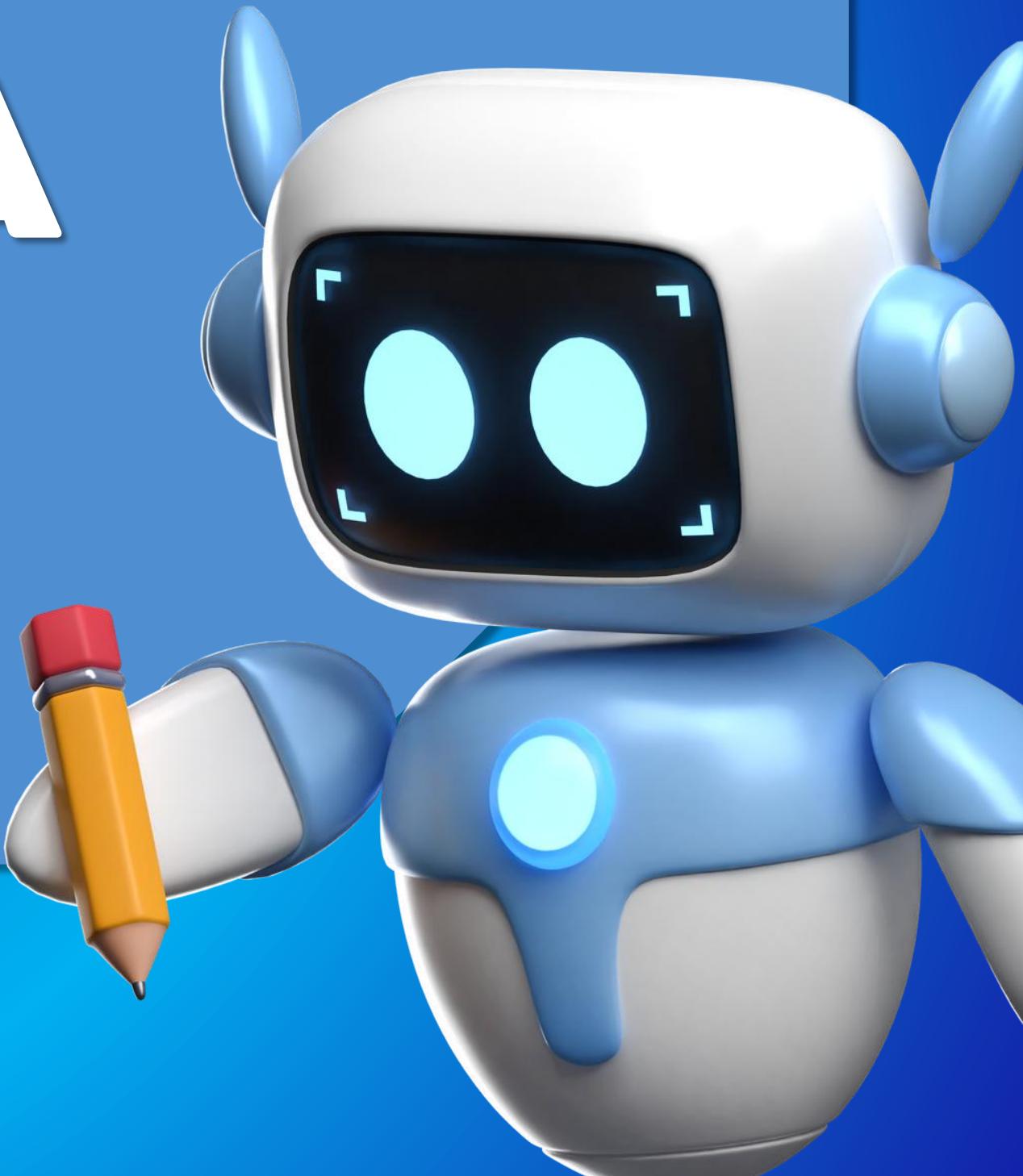
Podemos usar `dir()` para encontrar las "capacidades" de nuestra clase recién creada.

```
$ python party.py  
Type <class '__main__.PartyAnimal'>  
Dir ['__class__', ..., 'party', 'x']
```



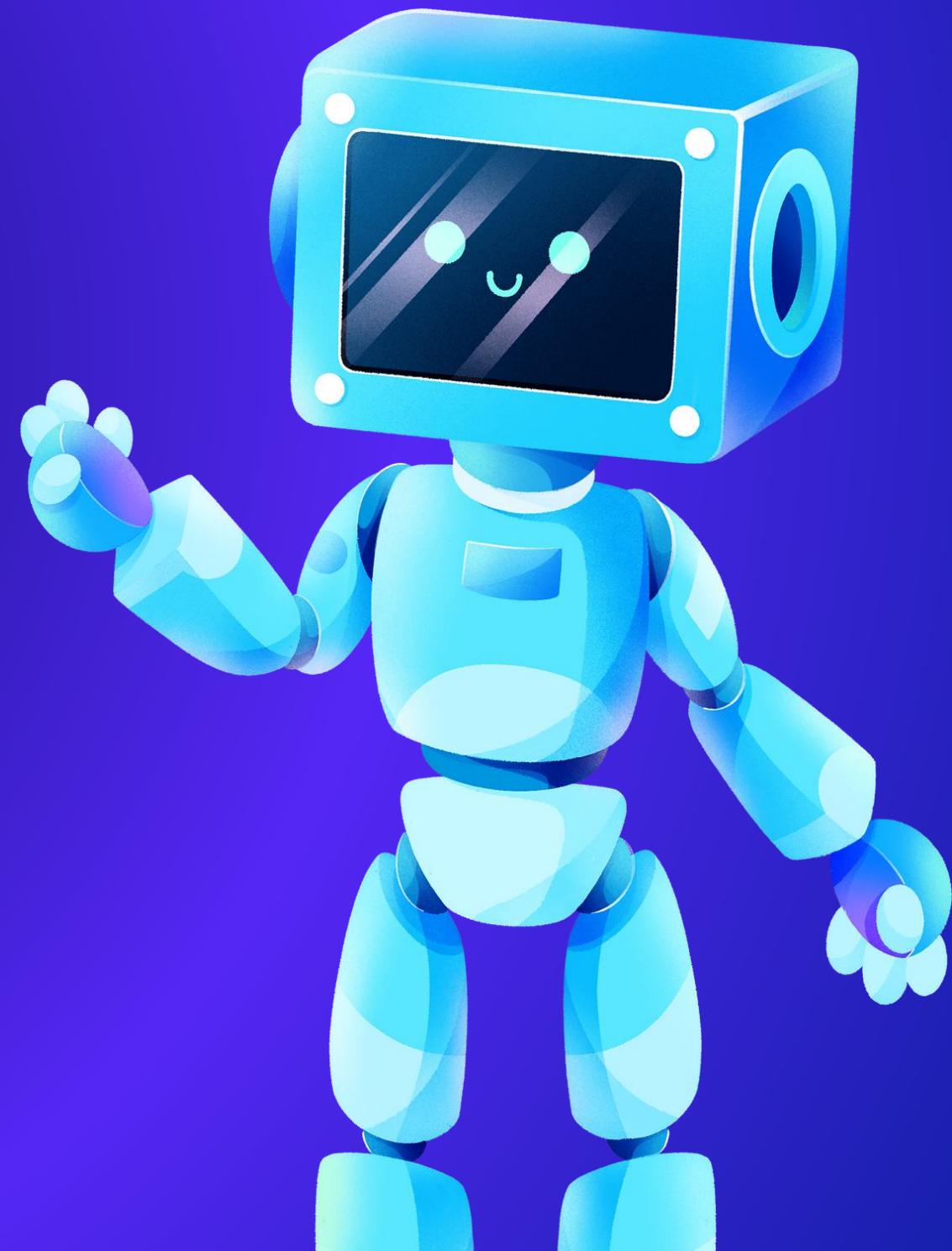


CICLO DE VIDA DEL OBJETO



CICLO DE VIDA DEL OBJETO

- Los objetos se crean, utilizan y descartan
- Tenemos bloques especiales de código (métodos) que se llaman:
 - En el momento de la creación (**constructor**).
 - En el momento de la destrucción (**destructor**).
- Los constructores se usan mucho
- Los destructores rara vez se usan



CONSTRUCTOR y DESTRUCTOR

El propósito principal del constructor es configurar algunas variables de instancia para que tengan los valores iniciales adecuados cuando se crea el objeto

```
class PartyAnimal:  
    x = 0  
  
    def __init__(self):  
        print("Estoy construido")  
  
    def party(self) :  
        self.x = self.x + 1  
        print('Hasta ahora',self.x)  
  
    def __del__(self):  
        print("Estoy destruido", self.x)  
  
an = PartyAnimal()  
an.party()  
an.party()  
an = 42  
print('an contiene',an)
```

```
$ python party2.py  
Estoy construido
```

```
Hasta ahora 1  
Hasta ahora 2
```

```
Estoy destruido 2  
an contiene 42
```

El constructor y el destructor son opcionales. El constructor se utiliza normalmente para configurar variables.

El destructor rara vez se usa.

MUCHAS INSTANCIAS

En programación orientada a objetos, un constructor en una clase es un bloque especial de sentencias llamadas cuando se crea un objeto.

Podemos crear muchos objetos: la clase es la plantilla para el objeto

Podemos almacenar cada objeto distinto en su propia variable

Llamamos a esto tener varias instancias de la misma clase.

Cada instancia tiene su propia copia de las variables de instancia

MUCHAS INSTANCIAS

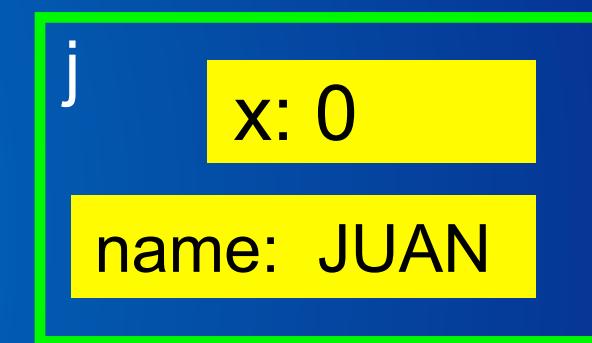
```
class PartyAnimal:  
    x = 0  
    name = ""  
    def __init__(self, z):  
        self.name = z  
        print(self.name, "construido")  
  
    def party(self) :  
        self.x = self.x + 1  
        print(self.name, "recuento", self.x)  
  
s = PartyAnimal("OMAR")  
j = PartyAnimal("JUAN")  
  
s.party()  
j.party()  
s.party()
```

Los constructores pueden tener parámetros adicionales. Estos se pueden utilizar para configurar variables de instancia para la instancia particular de la clase (es decir, para el objeto particular).

MUCHAS INSTANCIAS

```
class PartyAnimal:  
    x = 0  
    name = ""  
    def __init__(self, z):  
        self.name = z  
        print(self.name,"construido")  
  
    def party(self) :  
        self.x = self.x + 1  
        print(self.name,"recuento",self.x)  
  
s = PartyAnimal("OMAR")  
j = PartyAnimal("JUAN")  
  
s.party()  
j.party()  
s.party()
```

Tenemos dos
instancias
independientes



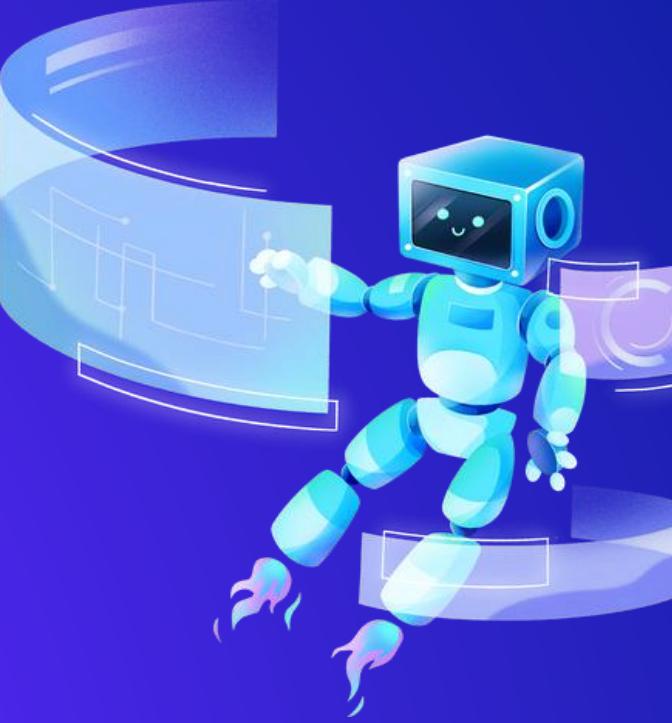
```
$ python instancias.py  
OMAR construido  
JUAN construido  
OMAR recuento 1  
JUAN recuento 1  
OMAR recuento 2
```

USANDO EL EDITOR



Usemos Python en nuestras maquinas

DESAFIO 1



```
1 # Convierte de Programación estructurada a POO
2
3 def calcular_area_triangulo(base, altura):
4     return (base * altura) / 2
5
6 def calcular_area_rectangulo(base, altura):
7     return base * altura
8
9 base_triangulo = 5
10 altura_triangulo = 8
11 area_triangulo = calcular_area_triangulo(base_triangulo, altura_triangulo)
12 print(f"Área del triángulo: {area_triangulo}")
13
14 base_rectangulo = 6
15 altura_rectangulo = 4
16 area_rectangulo = calcular_area_rectangulo(base_rectangulo, altura_rectangulo)
17 print(f"Área del rectángulo: {area_rectangulo}")
18
```

Ejercicio: Transforma este código en un conjunto de clases (Triangulo y Rectángulo) que tengan métodos para calcular su área.

DESAFIO 2

```
class Coche:  
    def __init__(self, marca, modelo):  
        self.marca = marca  
        self.modelo = modelo  
  
    def obtener_informacion(self):  
        print(f"Coche: {marca} {modelo}")  
  
mi_coche = Coche("Toyota", "Corolla")  
mi_coche.obtener_informacion()
```

Ejercicio: Identifica y corrige los errores en el código proporcionado.



MUCHAS GRACIAS POR SU ATENCIÓN



Centro de Graduados de la
Facultad de Ingeniería



Cooperativa de Trabajo
Tu Exito Profesional

