

pragma

Bi BANCO
INDUSTRIAL

Fundamentos de programación **en Python**

Módulo 1

Conceptos iniciales

Contenidos

¿Por qué elegir Python?

Introducción a Jupyter

Principales librerías en Python

Variables

Condicionales

Tipos de datos

Bucles

Ejercicios

01

Introducción



¿Por qué elegir Python?



Python es un lenguaje de programación versátil y puede utilizarse para una **amplia variedad** de aplicaciones, desde desarrollo web hasta análisis de datos, inteligencia artificial, automatización, scripting, entre otros. Algunas de sus ventajas son:

Facilidad de aprendizaje

Desarrollo rápido

Multiplataforma

Open Source

Gran comunidad

Interpretado no compilado

Documentación completa

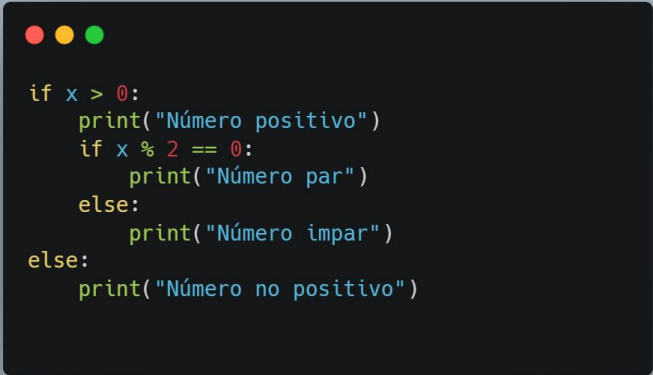
Características clave del lenguaje



Características clave del lenguaje

Indentación en Python

En Python, la indentación se refiere a la cantidad de espacios o tabulaciones al principio de una línea de código. Estos espacios indican la estructura y la jerarquía del código, definiendo bloques de código dentro de las estructuras de control, funciones, clases, etc.



```
if x > 0:
    print("Número positivo")
    if x % 2 == 0:
        print("Número par")
    else:
        print("Número impar")
else:
    print("Número no positivo")
```

Ventajas: Legibilidad – Consistencia – Reducción de errores.

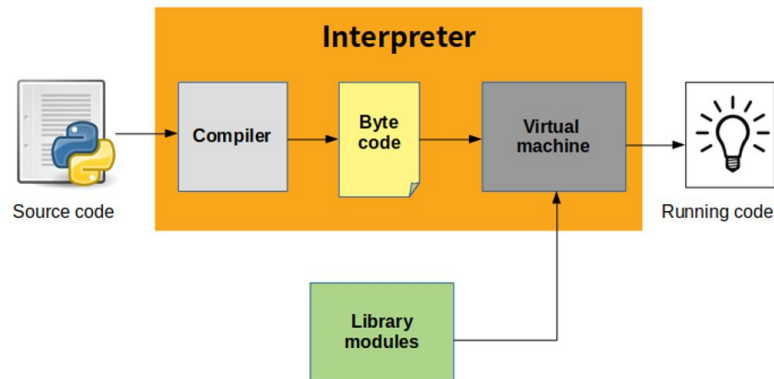
Características clave del lenguaje

Intérprete de Python

Un intérprete es un programa de computadora que lee, analiza y ejecuta instrucciones o código fuente escrito en un lenguaje de programación.

1. Código fuente
2. Compilación
3. Python virtual machine
4. Ejecución
5. Resultados

Esquema de funcionamiento



Fuente: [Medium](#)

Introducción a Jupyter

The image shows the JupyterLab interface with several components highlighted by red boxes and labeled with arrows:

- Barra de herramientas**: Points to the top menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help).
- Barra de Comandos**: Points to the toolbar below the menu bar.
- Selección de Kernel**: Points to the kernel selection dropdown in the top right, showing "Python 3 (ipykernel)".
- Explorador de archivos**: Points to the file browser on the left, showing a list of files in the "/ notebooks /" directory. The file "lorenz.py" is highlighted.
- Celda de Markdown**: Points to the first cell in the main editor, which contains the title "The Lorenz Differential Equations" and a paragraph of text.
- Celda de Código**: Points to the second cell in the main editor, which contains a code block starting with `%matplotlib inline` and `from ipywidgets import interactive, fixed`.

The main editor displays the Lorenz system of differential equations:

$$\dot{x} = \sigma(y - x)$$

Below the equations, there is an "Output View" showing a 3D plot of the Lorenz attractor. The plot has three sliders for parameters: sigma (10.00), beta (2.63), and rho (28.00).

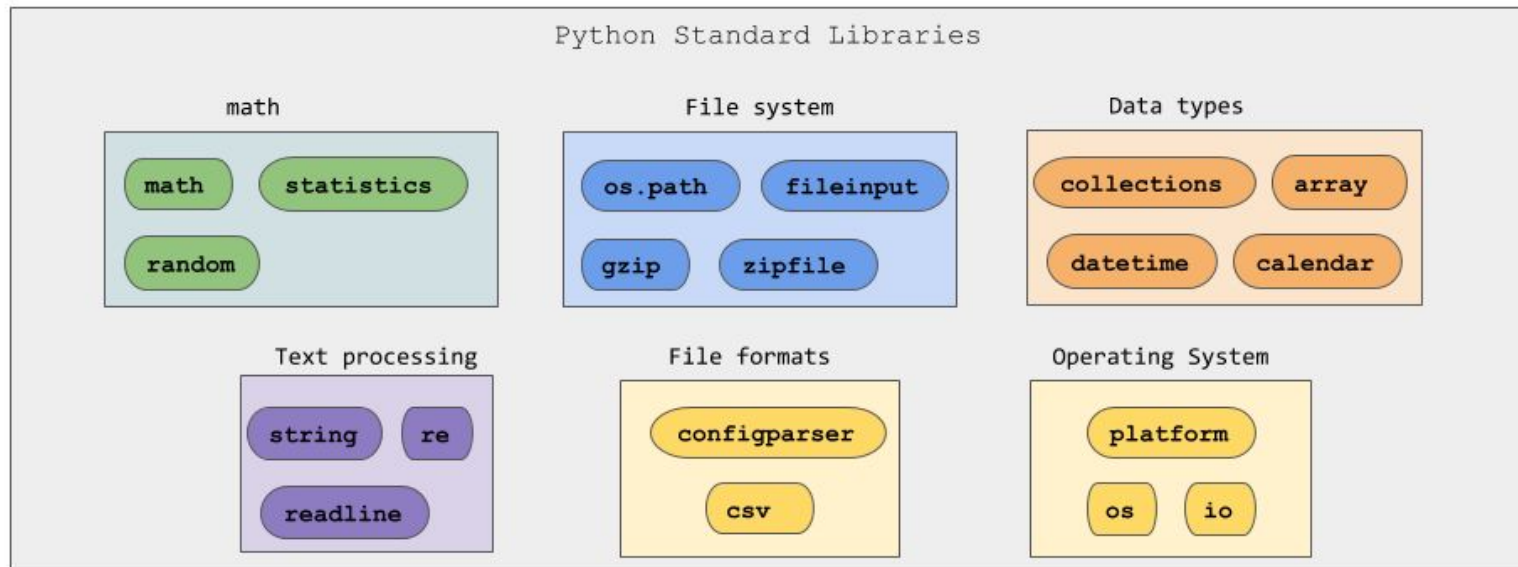
The "lorenz.py" file is open in the right pane, showing the following code:

```
1 from matplotlib import pyplot as plt
2 from mpl_toolkits.mplot3d import Axes3D
3 import numpy as np
4 from scipy import integrate
5
6 def solve_lorenz(sigma=10.0, beta=8./3, rho=28.0):
7     """Plot a solution to the Lorenz differential
8     equations."""
9
10    max_time = 4.0
11    N = 30
12
13    fig = plt.figure()
14    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
15    ax.axis('off')
```

Introducción a Jupyter

- Cada celda de un notebook puede contener código o markdown.
- Con markdown se puede mejorar la apariencia / legibilidad del notebook incluyendo, tablas, links, imágenes y formato a los textos.
- Para ejecutar una celda podemos dar clic en el botón play ► o usar las teclas Shift + Enter.
- Cuando una celda está en ejecución se marca con un asterisco.
- Cuando la celda termina de ejecutar se marca con un número en el orden de ejecución.
- Se puede ejecutar el notebook completo o a partir de una celda en particular.
- Las variables / comandos ejecutados en una celda, están disponibles en las demás celdas del notebook.
- Al reiniciar el kernel se eliminan todas las variables, datos, librerías de la memoria.

Principales librerías



Fuente: [Where are Python Libraries](#)

Principales librerías

Top 10 Python Libraries

**Pandas**

Data analysis and manipulation

**NumPy**

Mathematical functions

**Matplotlib**

Data visualisations

**SeaBorn**

Data visualisations

**Tensorflow**

Machine Learning

**Keras**

Deep Learning

**SciPy**

Scientific computing

**PyTorch**

Machine Learning

**Scrapy**

Web crawling

**SQLModel**

Interact with SQL databases

 DATA RUNDOWN**Django****Pyramid****Bottle****Tornado****Flask****Aiohttp****Pre-commit****Request****Numpy****Pandas****Pillow****Tensorflow****Scikit-learn****Pytorch****Opencv**

Variables y condicionales

Variables

```
# Asignación de variables
x = 1
print(x)
```

- Son referencias a objetos.
- Las variables y los objetos se almacenan en diferentes zonas de memoria.
- Las variables siempre referencian a objetos y nunca a otras variables.
- Los objetos sí pueden referenciar a otros objetos. Ejemplo: listas.

Condicionales

```
x = 1
if x > 0:
    print('Valor positivo')
else:
    print('Valor negativo')
```

[1] ✓ 0.0s

... Valor positivo

- Selección de una de varias alternativas en base a alguna condición.
- Indentación para estructurar el código
- Importante el símbolo ':'
- Se puede usar `elif` para añadir más condiciones.
- Se puede anidar condicionales.

Tipos de datos y operadores

Name	Type	Description
Integers	int	Whole numbers, such as: 3 300 200
Floating point	float	Numbers with a decimal point: 2.3 4.6 100.0
Strings	str	Ordered sequence of characters: "hello" 'Sammy' "2000" "楽しい"
Lists	list	Ordered sequence of objects: [10,"hello",200.3]
Dictionaries	dict	Unordered Key:Value pairs: {"mykey": "value" , "name": "Frankie"}
Tuples	tup	Ordered immutable sequence of objects: (10,"hello",200.3)
Sets	set	Unordered collection of unique objects: {"a","b"}
Booleans	bool	Logical value indicating True or False

Fuente: [Learn about Python 3 data types — numbers and strings | by Shawn Ren | Medium](#)

✓ Operadores aritméticos

Operador	Desc
a + b	Suma
a - b	Resta
a / b	División
a // b	División Entera
a % b	Modulo / Resto
a * b	Multiplicacion
a ** b	Exponenciación

Tipos de datos y operadores

✓ Operadores de comparación

Operador	Desc
a > b	Mayor
a < b	Menor
a == b	Igualdad
a != b	Desigualdad
a >= b	Mayor o Igual
a <= b	Menor o Igual

✓ Operadores Lógicos

Operador	Desc
a and b	True, si ambos son True
a or b	True, si alguno de los dos es True
not a	Negación

✓ Operadores de Asignación

Operador	Desc
=	Asignación
+=	Suma y asignación
-=	Resta y asignación
*=	Multiplicación y asignación
/=	División y asignación
%=	Módulo y asignación
//=	División entera y asignación
**=	Exponencial y asignación
&=	And y asignación
=	Or y asignación
^=	Xor y asignación
>>=	Despl. Derecha y asignación
<<=	DEspl. Izquierda y asignación

Iteración / Bucles

Bucle While

```
▷ ▾  
# Ejemplo: Mostrar los primeros 3 objetos de una lista  
  
indice = 0  
numeros = [9, 4, 7, 1, 2]  
while indice < 3:  
    print(numeros[indice])  
    indice += 1  
[ ]
```

- Repetición de un bloque de código hasta que se deje cumplir una expresión (es decir, hasta que una condición evalúa a **False**).
- Si la condición evalúa a **False** desde el principio, el bloque de código nunca se ejecuta.
- Cuidado con los bucles infinitos.

Bucle For

```
▷ ▾  
peliculas = [23, 'Avatar', 'Star Wars']  
for pelicula in peliculas:  
    print(pelicula)  
[ ]
```

- Permite recorrer los items de una **secuencia** o un objeto **iterable**.
- Funciona en strings, listas, tuplas, etc.

Ejercicios

1. Escribir un programa que pida al usuario su peso (en kg) y estatura (en metros), calcule el índice de masa corporal y lo almacene en una variable, y muestre por pantalla la frase Tu índice de masa corporal es <imc> donde <imc> es el índice de masa corporal calculado redondeado con dos decimales.
2. Escribir un programa que pregunte al usuario una cantidad a invertir, el interés anual y el número de años, y muestre por pantalla el capital obtenido en la inversión.
3. Escribir un programa que pida al usuario un número entero positivo y muestre por pantalla todos los números impares desde 1 hasta ese número separados por comas.
4. Escribir un programa que pida al usuario un número entero y muestre por pantalla un triángulo rectángulo como el de más abajo, de altura el número introducido.

```
*  
**  
***  
****  
*****
```

5. Escribir un programa que almacene el abecedario en una lista, elimine de la lista las letras que ocupen posiciones múltiplos de 3, y muestre por pantalla la lista resultante.

Módulo 1

Repasemos los conceptos
clave

Quiz time!



<https://kahoot.it/>

Conceptos Importantes

Variables

X = 1

Tipado dinámico, el intérprete asigna el tipo de dato en tiempo de ejecución.

Condicionales

```
if x > 0:  
    print("Positivo")  
else:  
    print("Negativo")
```

Bucle For

```
for n in range(1, 10):  
    print(f"El numero es: {n}")
```

Tipos de datos

int: Entero **float**: Punto flotante **str**: String

list: Lista **dict**: Diccionario **tup**: tupla

set: Conjunto **bool**: Boleano

Operadores

$a > b$: Mayor que | $a < b$: Menor que

$a == b$: Igual a | $a != b$: Diferente de

$a >= b$: Mayor o igual | $a <= b$: Menor o igual

Imprimir / Leer datos

```
print("Esto es un mensaje en consola")
```

```
my_variable = input("Ingresa un valor")
```

```
str.format() | f"dar formato a {my_variable}"
```

Conceptos Importantes

Acceso a ficheros

```
with open('VariasLineas.txt') as my_file:
    for linea in my_file:
        print(linea, end= ' ')
```

```
import csv
```

```
with open("res/tabla_operaciones.csv") as fichero:
    data_reader = csv.reader(fichero, delimiter=',')
    for linea in data_reader:
        print(linea[0] + ' ---- ' + linea[1])
```

Comentarios

```
# Este es un comentario
```

```
"""
```

```
Este es un comentario multilínea
```

```
"""
```

Módulos / Librerías

```
import pandas as pd
```

```
from my_module import *
```

Docstrings

```
def funcion_de_prueba():
```

```
    """
```

```
        Esta es la documentación de
        la función de prueba
```

```
    """
```

```
    pass
```

```
help(funcion_de_prueba)
```

Cheat Sheet



Python Basics

Python Cheat Sheet for Beginners

Learn Python online at www.DataCamp.com

> How to use this cheat sheet

Python is the most popular programming language in data science. It is easy to learn and comes with a wide array of powerful libraries for data analysis. This cheat sheet provides beginners and intermediate users a guide to starting using python. Use it to jump-start your journey with python. If you want more detailed Python cheat sheets, check out the following cheat sheets below:



> Accessing help and getting object types

```
1 * 1 # Everything after the hash symbol is ignored by Python
help(max) # Display the documentation for the max function
type('a') # Get the type of an object - this returns str
```

> Importing packages

Python packages are a collection of useful tools developed by the open-source community. They extend the capabilities of the python language. To install a new package (for example, pandas), you can go to your command prompt and type in `pip install pandas`. Once a package is installed, you can import it as follows.

```
import pandas # Import a package without an alias
import pandas as pd # Import a package with an alias
from pandas import DataFrame # Import an object from a package
```

Fuente: [Python Cheat Sheet for Beginners](#)

> Getting started with lists

A list is an ordered and changeable sequence of elements. It can hold integers, characters, floats, strings, and even objects.

Creating lists

```
# Create Lists with [], elements separated by commas
x = [1, 3, 2]
```

List functions and methods

```
x.sorted(x) # Return a sorted copy of the list e.g., [1,2,3]
x.sort() # Sorts the list in-place (replaces x)
reversed(x) # Reverse the order of elements in x e.g., [2,3,1]
x.reverse() # Reverse the list in-place
x.count(2) # Count the number of element 2 in the list
```

Selecting list elements

Python lists are zero-indexed (the first element has index 0). For ranges, the first element is included but the last is not.

```
# Define the list
x = ['a', 'b', 'c', 'd', 'e']
x[1:] # Select 1st (inclusive) to 3rd (exclusive)
x[0] # Select the 0th element in the list
x[2:] # Select the 2nd to the end
x[-1] # Select the last element in the list
x[:3] # Select 0th to 3rd (exclusive)
```

Concatenating lists

```
# Define the x and y lists
x = [1, 3, 6]
y = [10, 15, 21]
x + y # Returns [1, 3, 6, 10, 15, 21]
3 * x # Returns [1, 3, 6, 1, 3, 6, 1, 3, 6, 1, 3, 6]
```

> Getting started with dictionaries

A dictionary stores data values in key-value pairs. That is, unlike lists which are indexed by position, dictionaries are indexed by their keys, the names of which must be unique.

Creating dictionaries

```
# Create a dictionary with {}
{'a': 1, 'b': 4, 'c': 9}
```

Dictionary functions and methods

```
x = {'a': 1, 'b': 2, 'c': 3} # Define the x dictionary
```



Módulo 2

POO y Ambientes virtuales

Contenidos

¿Qué es la POO?

Principios de la POO

Clases y objetos

Constructor de una clase

Métodos de una clase

Encapsulamiento

Herencia

Manejo de excepciones

Contenidos

Manejo de dependencias en Python (Poetry)

Introducción a Pipenv y Venv

¿Cómo instalar versiones específicas de una librería?

Reto #1

Reto #2

01

Programación orientada a objetos – POO



Introducción a la programación orientada a objetos

¿Qué es la POO?

La programación orientada a objetos es un **paradigma de programación** que se basa en el concepto de objetos, que son entidades que tienen atributos (datos) y métodos (comportamientos).

¿Por qué es importante?

Su importancia radica en que facilita el desarrollo de software a través de la **reutilización** de código, la modularidad, la **flexibilidad** y capacidad de mantener el código entre otras características.

Encapsulamiento

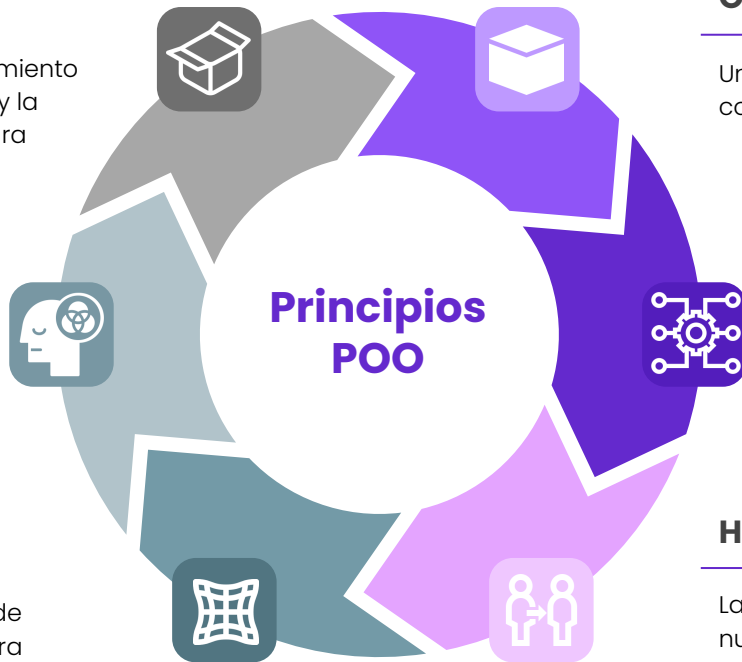
El encapsulamiento implica el ocultamiento de los detalles internos de una clase y la exposición de una interfaz pública para interactuar con el objeto.

Abstracción

La abstracción implica la simplificación y representación de los elementos esenciales de un sistema mientras se ocultan los detalles innecesarios.

Polimorfismo

El polimorfismo permite que objetos de diferentes clases se utilicen de manera uniforme.



Objeto

Un objeto es una instancia concreta de una clase.

Clase

Una clase es una plantilla o un modelo que define las características y comportamientos comunes de un grupo de objetos.

Herencia

La herencia permite la creación de nuevas clases basadas en clases existentes.

Clases y objetos

Veamos un ejemplo de cómo se define e instancia una clase en Python. Debemos prestar especial atención a los métodos, el constructor, y las propiedades de la clase.



Clase

Instancia



Objeto

Clases y objetos

¡ Importante !

- Por convención en python las clases se nombran con la **primera letra en mayúsculas** mientras que los métodos en minúsculas y separados con guión bajo (**snake case**).
- Para instanciar una clase, basta con usar la notación **variable = Clase**(arg1, arg2, ..., argn), donde arg corresponde a los argumentos que se pasan al método constructor.
- La palabra reservada **self** es una manera de referirse a la propia clase. Es similar al término **this** en otros lenguajes de programación.

```
class Casa:
    # Constructor
    def __init__(self, color, num_habitaciones):
        self.color = color
        self.num_habitaciones = num_habitaciones

    # Método para mostrar información de la casa
    def mostrar_informacion(self):
        print(f"Casa de color {self.color} con {self.num_habitaciones} habitaciones.")

    # Método para cambiar el color de la casa
    def pintar(self, nuevo_color):
        print(f"Cambiando el color de {self.color} a {nuevo_color}.")
        self.color = nuevo_color

# Instanciación de objetos (instancias de la clase Casa)
casa1 = Casa(color="Rojo", num_habitaciones=3)
casa2 = Casa(color="Azul", num_habitaciones=2)

# Acceso a propiedades y llamada a métodos
casa1.mostrar_informacion()
casa2.mostrar_informacion()

casa1.pintar("Verde")

# Mostrar información actualizada después de cambiar el color
casa1.mostrar_informacion()
```

Constructor y atributos de clase

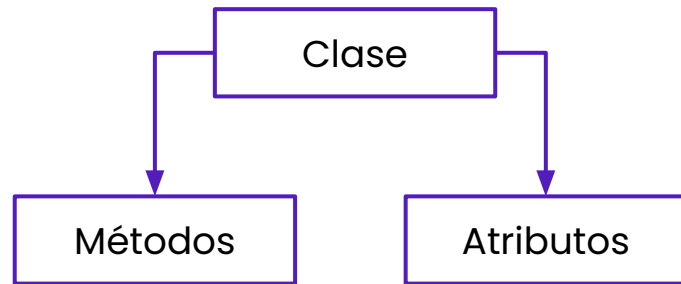
El constructor es un método especial de la clase que se encarga de asignar un **valor inicial o por defecto** a algunos atributos de la clase cuando esta es instanciada.

```
class Persona():
    def __init__(self, dni, nombre, apellido, edad):
        self.dni = dni
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad

persona = Persona("11111111A", "Ana", "López", 28)

print(persona.dni)
print(persona.nombre)
print(persona.apellido)
print(persona.edad)
```

Debemos prestar especial atención al nombre que tiene este método ya que debe llamarse **`__init__`**



1. Declarar la clase usando la palabra reservada **Class**.
2. Crear una instancia de la clase.
3. Acceder a los métodos y atributos de la clase usando la notación **objeto.abributo**
4. A diferencia de los atributos los métodos requieren usar paréntesis al final para ser invocados: **objeto.metodo()**

Funciones y métodos

Funciones

```
# Primero, definición.
def crear_diccionario(keys, values):
    return dict(zip(keys, values))

# Después, invocación.
nombres = ['Alfred', 'James', 'Peter', 'Harvey']
edades = [87, 43, 19, 16]
usuarios = crear_diccionario(nombres, edades)
print(usuarios)
```

Python

- Son una manera de agrupar un conjunto de instrucciones que pueden ser reutilizadas durante el flujo del programa.
- Las funciones deben ir identificadas por un nombre.
- Se debe usar el nombre para invocar (ejecutar) el código de la función.

Métodos

```
class Empleado:
    def __init__(self, nombre_empleado):
        self.nombre = nombre_empleado

    def presentar(self):
        print(f'Hola, me llamo {self.nombre}.')

empleado = Empleado('Carlos López')
empleado.presentar()
```

Python

- Los métodos son funciones que se definen dentro de una clase.
- Estos pueden o no acceder / modificar los atributos de la clase.
- Como cualquier función, los métodos de clase pueden recibir parámetros (argumentos) y retornar un valor (objeto).

Reto #1

Usted ha sido contratado por Banco Industrial para crear una aplicación que permita llevar un registro de los usuarios de la empresa. Para esto debe implementar una aplicación de línea de comandos que tenga las siguientes características:

1. Crear nuevos usuarios (Almacenar el nombre y el ID)
2. Consultar y mostrar en pantalla la información del usuario.

Para solucionar el reto se sugiere el uso de listas y diccionarios para almacenar la información en memoria. Para la consulta de datos y tener un menú con las opciones se sugiere el uso del bucle while.

Reto #2

En una zona de descanso de BI hay una máquina expendedora que está presentando fallas, no entrega los productos o devuelve una cantidad de dinero incorrecta. La empresa le ha asignado la misión de reparar el software de la máquina para que cumpla con las siguientes características:

- La máquina solo puede recibir monedas de 100, 200, 500 y 1000, en caso de insertar otra moneda, dinero insuficiente o un código de producto incorrecto se debe mostrar un mensaje de error y retornar el dinero.
- La máquina solo puede recibir una moneda a la vez.
- Cada producto de la máquina tiene un código único que lo identifica, para comprar un producto se debe introducir la moneda y el código de producto deseado.
- El programa debe retornar el nombre del producto. La máquina no retorna cambio.

Debe usar POO para resolver el problema, use una lista con diccionarios para almacenar la información de los productos que tiene la máquina. Una vez la máquina entrega el producto, finaliza la ejecución del programa.

Herencia en POO

```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

    def presentar(self):
        print(f'Hola, me llamo {self.nombre}.')

class Estudiante(Persona):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.asignaturas = []

    def matricular(self, asignatura):
        self.asignaturas.append(asignatura)

class Profesor(Persona):
    def __init__(self, nombre, salario):
        super().__init__(nombre)
        self.salario = salario

    def presentar(self):
        print(f'Hola, soy la profesora {self.nombre}.')

    def anyadir_sexenio(self):
        self.salario += 100

persona = Persona("Ana López")
persona.presentar()
```

Python

¿Qué es la herencia?

- La herencia hace referencia a la capacidad de una clase para adquirir **datos** y **comportamientos** de la clase padre.
- A la clase de la que se heredan estos comportamientos se le denomina clase padre o **superclase**.
- A la clase que hereda se le conoce como clase hija o **subclase**.

En el ejemplo, la subclase Estudiante **hereda los métodos y atributos** de la clase Persona. Al crear una instancia de Estudiante, es posible acceder al método presentar, a pesar de que este no está de manera explícita en su definición.

La función **super()** de python permite llamar métodos / atributos de la superclase desde la subclase.

Encapsulamiento

```
class Carro:
    def __init__(self, marca, modelo, precio):
        self.__marca = marca
        self.__modelo = modelo
        self.__precio = precio

    def get_precio(self):
        return self.__precio

mustang = Carro("Ford", "Mustang", 50000)

print(mustang.get_precio())
print(mustang.__precio) # Esta línea genera un error
```

[1] 0.3s Python



Manejo de excepciones

¿Por qué manejar las excepciones?



Permite controlar el flujo de ejecución.



Mejora la usabilidad del programa



Permite definir una estrategia de reintentos.



Evita bloqueos o indisponibilidad del sistema.



Cumplimiento de los requisitos de calidad.



Mejora la fiabilidad del programa.

```
try:
    archivo = open("archivo_no_existente.txt", "r")
except FileNotFoundError:
    print("Manejo del error.")
else:
    print("Este bloque se ejecuta si no hay error.")
finally:
    print("Este bloque siempre se ejecuta.")
```

Python

- En el bloque try se ingresa la instrucción que puede generar error.
- El bloque except define las instrucciones a ejecutar en caso de que se presente un error.
- Es una buena práctica definir explícitamente el error que se espera para controlarlo de manera adecuada.
- Pueden existir múltiples bloques except para capturar diferentes tipos de errores..

Ambientes virtuales

¿Para qué sirven?

Al momento de gestionar las dependencias, no basta con definir una **versión específica** de la librería, es posible que en nuestro sistema ya tengamos una versión instalada que pueda entrar en conflicto. Los ambientes virtuales solucionan este problema al crear un **ambiente de trabajo aislado**, únicamente con las librerías de nuestro proyecto y la versión de python requerida.



Pipenv

Una de las ventajas que ofrece pipenv frente a otras alternativas es que permite crear un ambiente virtual y gestionar las versiones de las librerías de manera simultánea.

```
> pipenv --python 3.11  
> pipenv install  
> pipenv shell  
> exit
```



Venv

Venv está presente por defecto en la instalación de python, para usarlo basta ejecutar el siguiente comando en consola:

```
> python -m venv /path/to/myenv  
> myenv\Scripts\activate  
> deactivate
```

Manejo de paquetes / librerías

¿Por que es necesaria la gestión de dependencias?



Permite replicar los experimentos.



Evita el conflicto entre versiones.



Facilita compartir el código.



Evita configuraciones manuales del entorno de trabajo.



Evita errores en la configuración de otros proyectos.



Poetry

```
%pip install poetry
%poetry init
|
%poetry install
```



Pipenv

```
%pip install pipenv
%pipenv --python 3.11
%pipenv install --dev
|
%pipenv shell
```

Instalar versiones específicas de librerías

Version specifiers:

~= (Compatible release clause): Acepta la versión especificada y cualquier versión posterior compatible con ella.

== (Version matching clause): Exige solo la versión exacta indicada.

!= (Version exclusion clause): Excluye la versión indicada.

<=, >= (Inclusive ordered comparison clauses): Indican un rango de versiones aceptables.

<, > (Exclusive ordered comparison clauses): Indican un rango de versiones, pero sin incluir la versión especificada.

Poetry

```
>> poetry add pandas
```

pyproject.toml

Pipenv

```
>> pipenv install pandas==2.2.0
```

Pipfile

Pip

```
>> pip install pandas==2.2.0
```

requirements.txt

Módulo 3

Ciencia de datos con Python

Contenidos

Estructuras de datos en python

Manipulación de datos

Análisis exploratorio (EDA)

Estadística descriptiva con Python

Visualización de datos

Reto #3

Glosario de términos

- **Atributo:** Una propiedad o característica de un objeto que almacena información sobre el objeto.
- **Método:** Una función asociada a una clase u objeto que define su comportamiento o acción.
- **Instancia:** Un objeto específico creado a partir de una clase.
- **Constructor:** Un método especial en una clase que se llama automáticamente cuando se crea una nueva instancia de la clase. Se utiliza para inicializar los atributos del objeto.
- **Destructor:** Un método especial que se llama automáticamente cuando un objeto se destruye o sale de alcance. Se utiliza para realizar operaciones de limpieza.
- **Interfaz:** El conjunto de métodos y propiedades que define cómo interactuar con un objeto o una clase. Puede ser considerado como un contrato que indica qué operaciones están disponibles.
- **Polimorfismo de Sobrecarga:** La capacidad de una clase para tener múltiples métodos con el mismo nombre pero con diferentes parámetros. Permite a un objeto realizar diferentes acciones dependiendo de los argumentos recibidos.
- **Polimorfismo de Anulación:** La capacidad de una clase derivada para proporcionar una implementación específica de un método que ya está definido en la clase base.
- **Clase Abstracta:** Una clase que no puede ser instanciada por sí misma y generalmente contiene métodos abstractos. Sirve como una plantilla para clases derivadas.

Glosario de términos

- **Método Abstracto:** Un método en una clase abstracta que no tiene implementación en la clase base y debe ser implementado por cualquier clase derivada.
- **Composición:** Un concepto donde un objeto contiene otros objetos como parte de su estructura interna.
- **Herencia Múltiple:** La capacidad de una clase para heredar atributos y métodos de más de una clase base.
- **Encapsulamiento de Datos:** La ocultación de los detalles internos de la implementación de una clase, permitiendo el acceso controlado a través de métodos públicos.
- **Sobrecarga de Operadores:** La capacidad de definir múltiples versiones de un operador para comportarse de manera diferente dependiendo de los tipos de operandos.
- **Herencia de Implementación:** La reutilización de la implementación de una clase base en una clase derivada.
- **Sobreescritura:** La acción de proporcionar una implementación específica de un método que ya está definido en la clase base.
- **Visibilidad (Public, Private, Protected):** Define el alcance de acceso a los atributos y métodos. Public es accesible desde cualquier parte, Private sólo es accesible desde la propia clase, y Protected es accesible desde la propia clase y sus clases derivadas.
- **Mixin:** Una técnica donde una clase proporciona métodos y atributos adicionales que pueden ser utilizados por otras clases sin la necesidad de heredar de ella.

Mejoramos
la vida de la gente
transformando
empresas

pragma

www.pragma.co



Carrera 42 # 5 Sur 47
Edificio SELF - Piso 16
Medellín, Colombia
t. (323) 563 9223

o o o Keep moving

