PYTHON BASICO

Como vamos a trabajar y requisitos necesarios

Voy a utilizar Github para los códigos que vayamos realizando en el equipo además de este documento que lo tendremos también ahí.

1. Instalar Python

Texto

El contenido generado por IA puede ser incorrecto.

1. Visual Studio Code. Es un IDE para trabajar con cualquier lenguaje.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Python es un lenguaje de programación que se comunica con TODO.

Python se ha hecho popular por Hacking y Big Data

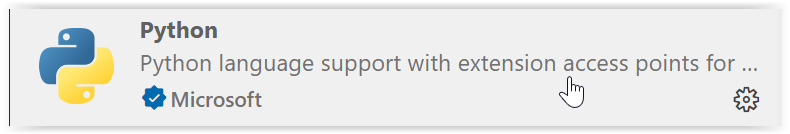
Dentro de Python tenemos multitud de características:

* Hacking: Como Python puede comunicar con cualquier dispositivo y sistema, se ha hecho popular para los programas de Hacking
* Big Data: Estos conceptos los utilizaban los matemáticos y decidieron este lenguaje porque es muy intuitivo y fácil de aprender.
* Django: Es un framework para generar páginas Web con código Python
* Fast Api/Flask: Es un Framework para generar apis con lenguaje Python
* IA: Crear modelos/agentes de inteligencia artificial
* Jupyter: Es un Framework para análisis de datos y generar gráficos

Visual Studio Code nos permite trabajar con cualquier entorno de los que hemos escrito.

Mediante extensiones de VS Code, podemos acceder a cualquier característica de Python o cualquier lenguaje.

Vamos a instalar las extensiones de Python/Jupyter dentro de VS Code.



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Todas las extensiones de ficheros de Python (mentira) terminan en **.py**

Para trabajar, vamos a crear una carpeta en nuestro ordenador y, a partir de ahí tendremos nuestros códigos.

Dependiendo del tipo de proyecto, se generan más carpetas de trabajo.

Abrimos esta carpeta desde Visual Studio Code

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.



**Importante en el lenguaje:**

* **Diferencia mayúsculas de minúsculas**
* **Las tabulaciones son básicas, son las que marcan el programa**
* **Las variables (mentira) no tienen tipado, simplemente se escriben sin declara su tipo. Se genera su tipo al almacenar un valor en su interior**

Vamos a trabajar de forma consecutiva al nombrar los programas.

Llamaremos a los programas **pythonXXdescripcion.py**

Vamos a comenzar viendo las variables y algún tipo de conversión.

Creamos un nuevo programa llamado **python01variables.py**

**print("Ejemplo de variables")**

**numero = 14**

**texto = "Primer Python"**

**#Comentarios en Python**

**#Comentar dentro de VS Code**

**#Comentar: CONTROL + K + C**

**#Descomentar: CONTROL + K + U**

**print(numero)**

**print(texto)**

**#El simbolo + sirve para concatenar y también para sumar**

**#Verifica el tipo de dato**

**print("Texto: " + texto)**

**#print("Numero: " + numero)**

**#También podemos utilizar la coma para concatenar**

**print("Numero: ", numero)**

**#print f nos permite concatenar múltiples variables en String sin**

**#importar el tipo de dato.**

**#Cada variable lógica debe ir entre llaves {}**

**#La letra f se escribe ANTES del String y fuera del string**

**print(*f*"El texto es {texto} y el número es {numero}")**

**#Dentro de las variables primitivas: str, int, boolean, float**

**#podemos convertir mediante funciones de Python**

**#str(variable): Convierte un tipo a String**

**#int(variable): Convierte un tipo a Entero**

**#float(variable): Convierte un tipo a Decimal**

**print("Numero: " + *str*(numero))**

CONDICIONALES

Un condicional permite estructurar un lenguaje según preguntas que vayamos realizando en el programa.

El código no será lineal.

Los condicionales están delimitados mediante Tabulaciones en sus instrucciones.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Operadores de comparación en Python:

> Mayor

< Menor

>= Mayor o igual

<= Menor o igual

== Igual

!= Distinto

Vamos a realizar un simple programa para visualizar la sintaxis y comparar si un número es positivo, negativo o cero.

Pediremos un número al usuario. Para pedir datos al usuario en este entorno se utiliza la función **input()**. Dicha función nos devuelve siempre un STRING, así que tendremos que convertir si deseamos tener un número.

Creamos un nuevo programa llamado **python02condicionales.py**

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Tenemos la posibilidad de incluir una condición ELSE

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Perfectamente se pueden **anidar** todas las instrucciones que deseemos, por ejemplo, si el número es cero.

Texto

El contenido generado por IA puede ser incorrecto.

Dentro del lenguaje, tenemos otra forma de preguntar dentro de un mismo IF si la pregunta tiene que ver entre sí.

Con la instrucción **ELIF** podemos seguir haciendo preguntas dentro de un mismo IF

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**DEPURACION DE CODIGO**

La depuración de código nos permite visualizar las variables dinámicamente en nuestro programa y nos permite averiguar que va sucediendo paso a paso.

En cualquier programa, tenemos tres tipos de errores:

1. **Errores de compilación:** Nos equivocamos al escribir y tenemos mal la sintaxis.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

1. **Errores en ejecución:** El programa compila y ejecuta, pero se detiene en algún momento por alguna instrucción que no hemos controlado, por ejemplo, en el programa anterior, que escriba un texto en lugar de un número.

Texto

El contenido generado por IA puede ser incorrecto.

1. **Errores lógicos:** Mi programa funciona, todo es correcto, ningún error, pero NO hace lo que yo quiero, necesito saber qué está pasando dentro del código. Solución es utilizar Debug

Para poder depurar dentro de Python debemos crear un punto de interrupción a la izquierda de los números. En la línea que deseemos comenzar a depurar nuestro código.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Para ejecutar el programa en modo Debug, entramos en un icono a la izquierda de un bicho y pulsamos en **Run and Debug**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Cuando hablamos de condicionales, tenemos también los operadores relacionales para hacer más de una pregunta

Dos palabras clave:

* **and:** Todas las condiciones deben cumplirse
* **or:** Cualquier condición entrará dentro del IF.

Para probar esta funcionalidad vamos a realizar un ejemplo para averiguar el mayor, menor e intermedio de tres números que vamos a pedir al usuario.

Creamos un nuevo programa llamado **python03mayormenor.py**

Texto

El contenido generado por IA puede ser incorrecto.

**print("Operadores relacionales")**

**num1 = *int*(input("Numero 1: "))**

**num2 = *int*(input("Numero 2: "))**

**num3 = *int*(input("Numero 3: "))**

**mayor = 0**

**menor = 0**

**intermedio = 0**

**if (num1 >= num2 and num1 >= num3):**

**mayor = num1**

**elif (num2 >= num1 and num2 >= num3):**

**mayor = num2**

**else:**

**mayor = num3**

**if (num1 <= num2 and num1 <= num3):**

**menor = num1**

**elif (num2 <= num1 and num2 <= num3):**

**menor = num2**

**else:**

**menor = num3**

**suma = (num1 + num2 + num3)**

**intermedio = suma - mayor - menor**

**print(*f*"Mayor: {mayor}, Menor: {menor}, Intermedio: {intermedio}")**

**print("Fin de programa")**

**BUCLES**

Un bucle es una secuencia que se repite N veces en el código.

Tenemos dos tipos de bucles:

1. **while:** Bucle condicional que finaliza cuando se cumple una condición expresada en su declaración. Dentro del bucle, debemos realizar acciones para que la condición se cumpla y salga del bucle.

Sintaxis:

while (condicion == true):

#instrucciones

condicion = false

print(“Fin de programa”)

1. **for:** Bucles contador. Se repiten N veces en una secuencia y finalizan cuando el bucle ha llegado al final del contador. Se declara la propia variable contador dentro de la definición del bucle.

**Por defecto, un bucle for siempre comienza en CERO**

Sintaxis:

for **contador** in range(número final):

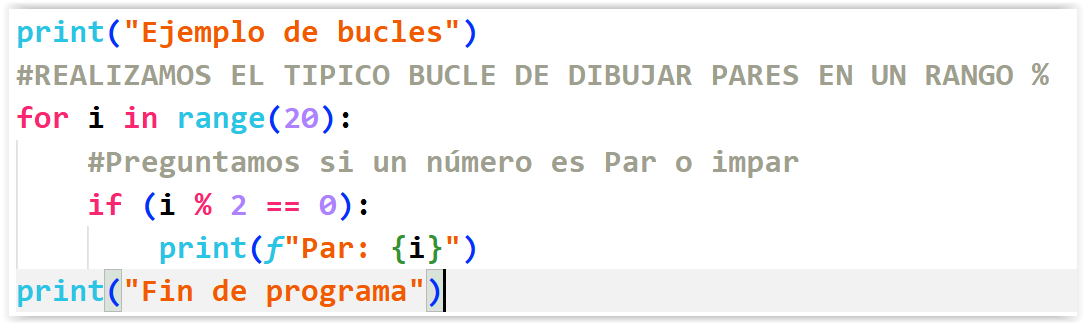
#instrucciones

Tenemos otro tipo de sintaxis en el que podemos indicar el inicio del bucle, además del final.

for **contador** in range(**inicio**, final):

#instrucciones

Creamos un nuevo programa llamado **python04bucles.py**



Vamos a crear un bucle para comprobar la Conjetura de Collatz

La conjetura de Collatz indica lo siguiente:

Todo número entero positivo será siempre 1 siguiendo estas instrucciones:

* Si el número es Par, dividimos entre 2
* Si el número es Impar, multiplicamos por 3 y sumamos 1

6, 3, 10, 5, 16, 8, 4, 2, 1

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**CLASE STRING PYTHON**

Los objetos, al ser declarados con algún valor, como String, contienen una serie de métodos y propiedades.

Dentro de Python existe el concepto de clases. Un objeto al ser declarado con una clase determinada contendrá métodos y propiedades de la clase.

Existen clases que podemos utilizar simplemente, como la clase String dentro del programa o, existen clases que debemos importar para utilizar en nuestro programa, por ejemplo, la clase **math**

Por último, existen clases que no están dentro del compilador de Python y que tendremos que instalarlas para utilizarlas, por ejemplo, clases para generar gráficos o clases para conectar a bases de datos.

Un String es un conjunto de char/letras.

Todo conjunto se puede medir su longitud mediante una función: **len()**

Un String diferencia mayúsculas de minúsculas en sus caracteres.

texto = “Lenguaje Python”

**len(texto)** 🡪 15

Dentro de los conjuntos podemos acceder de forma individual a cada elemento dentro del conjunto, en esta teoría, podemos acceder a cada letra mediante su índice.

Todo conjunto comienza en CERO

texto[0] 🡪 L

texto[1] 🡪 e

Tenemos una serie de métodos para trabajar con estos objetos string:

* lower(): Convierte a minúscula un texto
* upper(): Convierte a mayúscula un texto
* find(“Contenido a buscar”): Busca el contenido dentro del texto y nos devuelve su posición. Si no lo encuentra, nos devuelve -1
* rfind(“Contenido a buscar”): Busca el contenido dentro del texto y nos devuelve su posición. Si no encuentra, nos devuelve -1. Comienza a buscar desde el final

Dentro de Python tenemos POO y eso nos permite tener sobrecarga en algunos métodos/funciones

* find(“Contenido a buscar”, índice): Busca el contenido dentro del texto y nos devuelve su posición comenzando a buscar desde el índice. Si no lo encuentra, nos devuelve -1
* startswith(“contenido”): Indica si el texto comienza con el contenido
* endswith(“contenido”): Indica si el texto finaliza con el contenido
* replace(“antiguo”, “nuevo”): Sustituye el contenido del texto antiguo por el nuevo
* count(“texto”): Nos devuelve el número de veces que aparece el texto

Tenemos métodos para preguntar por el tipo de contenido del string:

* isdigit(): Indica si el contenido del texto son números
* isalpha(): Indica si el contenido del texto son letras
* isalnum(): Indica si el contenido está formado por letras y números
* isupper(): Indica si un texto está en mayúsculas
* islower(): Indica si el texto está en minúscula

Dentro de Python, en los conjuntos, existe un término que se llama **SLICING** que nos permite extraer un subconjunto de un conjunto.

En nuestro ejemplo actual, nos permite extraer una subcadena del texto.

texto = “primer python”

Tenemos dos posibilidades:

1. Queremos la subcadena desde una posición en adelante Objeto[INDICE:]

texto[8:] 🡪 ython

1. Devolver una subcadena desde una posición inicial hasta una posición final.

Texto[0:2] 🡪 pri

Para tener un ejemplo de la clase String vamos a realizar una validación de email y no utilizaremos bucles, solamente los métodos de la clase string.

* Que el email contenga una @
* Que el email contenga un punto
* @ ni al inicio ni al final
* Solamente una @ en el mail
* Un punto después de la @
* Dominio de 2 a 3 caracteres

Creamos un nuevo programa llamado **python05validaremail.py**

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**print("Validacion de email")**

**email = input("Introduzca email: ")**

**# • Que el email contenga una @**

**# • Que el email contenga un punto**

**# • @ ni al inicio ni al final**

**# • Solamente una @ en el mail**

**# • Un punto después de la @**

**# • Dominio de 2 a 3 caracteres**

**if (email.find("@") == -1):**

**print("No existe @")**

**elif (email.count(".") == 0):**

**print("No existe punto")**

**elif (email.startswith("@") == True or email.endswith("@") == True):**

**print("@ al inicio o al final")**

**elif (email.count("@") > 1):**

**print("Solamente una @ en el mail")**

**elif (email.find("@") > email.rfind(".")):**

**print("Debe existir un punto despues de la @")**

**else:**

**#posicion del ultimo punto**

**ultimoPunto = email.rfind(".")**

**#Recuperamos el dominio a partir de la posicion**

**#del ultimo punto**

**#paco@gmail.com**

**#email[4:] -> @gmail.com**

**#email[0] -> p**

**#email[1] -> a**

**dominio = email[ultimoPunto + 1:]**

**if (len(dominio) >= 2 and len(dominio) <= 3):**

**print("Email correcto")**

**else:**

**print("El dominio debe ser de 2 a 3 caracteres")**

**print("Fin de programa")**

Vamos a realizar un ejemplo para recorrer un String numérico.

Pediremos al usuario un texto de solo números.

Mostraremos la suma de dichos números.

Recorreremos carácter a carácter el texto y lo iremos sumando.

1234 🡪 10

Creamos un nuevo programa llamado **python06sumarnumerosstring.py**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**print("Sumar numeros textos")**

**textoNumeros = input("Introduzca solo numeros: ")**

**suma = 0**

**#1234**

**#Realizamos un bucle para recorrer cada letra del texto String**

**for i in range(len(textoNumeros)):**

**#recuperamos cada letra de cada posicion textoNumero[posicion]**

**letra = textoNumeros[i] # "1"**

**#convertimos el caracter "1" a int 1**

**numero = *int*(letra)**

**suma = suma + numero**

**print(*f*"La suma de {textoNumeros} es {suma}")**

**print("Fin de programa")**

**METODOS/FUNCIONES DENTRO DEL LENGUAJE PYTHON**

Una función es un método también, en realidad son lo mismo, una secuencia de instrucciones.

Son herramientas que contiene el lenguaje Python y que podemos utilizar en nuestros programas.

* Un método es una acción que se realiza sobre un objeto. Por ejemplo, hemos utilizado el método **startswith** de la clase String. Lo hemos utilizado sobre una variable que hemos llamado **email**

La instrucción que hemos utilizado ha sido:

Primero el objeto:

**email =** [**paco@gmail.com**](mailto:paco@gmail.com)

Aplicamos el método al objeto email de la clase String

**email.startsWith(“@”)**

* Una función no utiliza clase ni objeto, simplemente se llama como herramienta del lenguaje, por ejemplo, hemos utilizado la función **len()** sin invocar ningún objeto ni nada

**len(**“texto”**)**

Un método/función nos permiten realizar código NO lineales, lo que quiere decir que podemos tener el programa separado en zonas, e incluso en diferentes programas de Python.

Un método también nos permite poder reutilizar nuestro código llamando múltiples veces al método sin escribir su contenido de nuevo.

Los métodos pueden estar en el mismo programa en el que estoy escribiendo o en programas Python separados.

Todo funciona mediante llamadas y declaraciones de los métodos.

Por ejemplo, pongamos que tenemos un método que nos valida un email.

**PROGRAMA PRINCIPAL MAIN.PY**

print(“Mi programa principal”)

print(“Introduzca email”)

email = input()

**validarEmail()**

print(“Fin de programa”)

#METODO PARA VALIDAR UN EMAIL

**validarEmail()** {

print(“Validando email”)

#NUESTRO CODIGO DE VALIDACION

}

Mi programa principal

Introduzca un email

[paco@gmail.com](mailto:paco@gmail.com)

**Validando email**

Fin de programa

Podríamos tener el código del método en otra clase distinta, por ejemplo:

**PROGRAMA PRINCIPAL MAIN.PY**

print(“Mi programa principal”)

print(“Introduzca email”)

email = input()

**métodos.validarEmail()**

print(“Fin de programa”)

**METODOS.PY**

#METODO PARA VALIDAR UN EMAIL

**validarEmail()** {

print(“Validando email”)

#NUESTRO CODIGO DE VALIDACION

}

Mi programa principal

Introduzca un email

[paco@gmail.com](mailto:paco@gmail.com)

**Validando email**

Fin de programa

Para crear métodos en Python se utiliza una palabra clave llamada **def** y, a continuación, el nombre del método.

La sintaxis en Python suele ser como en JavaScript/Java, la primera letra en minúscula y cada palabra del nombre del método en mayúscula cada letra inicial: **nombreMetodoNuevo()**

En Python, los métodos deberían estar al inicio de la clase.

Python NO es capaz de leer un método si todavía no lo reconoce en la llamada.

Por ahora, vamos a escribir nuestros métodos dentro de la misma clase **main.py**

Vamos a crear un programa llamado **python07metodos.py**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**#EN UN PROGRAMA LOS METODOS DEBEN ESTAR ANTES DE LA LLAMADA**

**#AL METODO**

***def* primerMetodo():**

**#ESTE CODIGO NUNCA SE EJECUTARA SI NO LO LLAMAMOS**

**print("Ejecutando metodo 1")**

***def* segundoMetodo():**

**print("Ejecutando metodo 2")**

**#---------------------------------------**

**#CODIGO PRINCIPAL DE NUESTRO PROGRAMA MAIN**

**print("Ejemplo de métodos")**

**#LLAMADA A LOS METODOS**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

**print("Fin de programa")**

## PARAMETROS EN LOS METODOS

Un método puede recibir parámetros, dichos parámetros NO son opcionales (por ahora), deben ser enviados de forma obligatoria en la llamada al método.

Los tipos de dato de los métodos no se definen (por ahora), simplemente se escribe el nombre de la variable que vamos a utilizar dentro del método.

Dicha variable tendrá ámbito de método, es decir, solamente la podemos utilizar dentro del método, no existe fuera

Un método puede contener múltiples parámetros, simplemente se separan mediante comas cada parámetro del método.

El orden de los parámetros en la llamada, será el orden de los parámetros en la declaración.

Ejemplo:

def metodoParametros(**nombre**):

print(“Su nombre es “ + **nombre**)

Petición al método

metodoParametros(14)

metodoParametros(“Alumno”)

Método con múltiples parámetros:

def metodosMultiplesParametros(**nombre, apellido, edad**):

print(“Su nombre es “ + **nombre** + “ “ + **apellidos**)

print(“Su edad es “, edad)

Llamada al método

metodosMultiplesParametros(“Paco”, “Garcia”, 44)

Vamos a probar esta teoría en el mismo programa en el que estamos de Python métodos

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**#EN UN PROGRAMA LOS METODOS DEBEN ESTAR ANTES DE LA LLAMADA**

**#AL METODO**

***def* primerMetodo():**

**#ESTE CODIGO NUNCA SE EJECUTARA SI NO LO LLAMAMOS**

**print("Ejecutando metodo 1")**

***def* segundoMetodo():**

**print("Ejecutando metodo 2")**

***def* saludar(*nombre*):**

**print(*f*"Hola en Python {nombre}")**

***def* despedirse(*nombre*, *dia*):**

**print(*f*"Hasta luego {nombre} en el día {dia}")**

**#---------------------------------------**

**#CODIGO PRINCIPAL DE NUESTRO PROGRAMA MAIN**

**print("Ejemplo de métodos")**

**saludar("Paco")**

**despedirse("Paco", "Martes")**

**#saludar("Paco")**

**#LLAMADA A LOS METODOS**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

**print("Fin de programa")**

Cuando hablamos de métodos, existen dos grandes tipos:

* **Métodos de acción:** Realizan una serie de acciones y ejecutan su código
* **Métodos return:** Ejecutan una serie de acciones y, al finalizar, devuelven un dato a la llamada al método.

Acción: **print()**

Return: **len()**

Para visualizarlo vamos a realizar en este mismo programa de métodos un ejemplo de sintaxis:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Vamos a crear un programa que realizará lo siguiente:

* Pediremos dos números al usuario
* Mostraremos un menú al usuario indicando si desea Sumar o Multiplicar

Tendremos tres métodos en la clase. mostrarMenu(), sumarNumeros(), multiplicarNumeros()

Realizaremos un bucle para que el usuario pueda seleccionar una opción del menú

Si el usuario no introduce números, le mostramos mensaje hasta que nos ofrezca un número

Para esta acción, vamos a crear un método que comprobará el dato introducido por el usuario.

Mediante **iddigit()** de la clase String comprobamos el dato y lo tendremos en el método hasta que nos ofrezca un número.

Creamos un nuevo programa llamado **python08metodoscalculadora.py**

***def* sumarNumeros(*num1*, *num2*):**

**return num1 + num2**

***def* multiplicarNumeros(*num1*, *num2*):**

**return num1 \* num2**

***def* mostrarMenu():**

**print("0.- Salir")**

**print("1.- Sumar")**

**print("2.- Multiplicar")**

**print("3.- Introduzca nuevos numeros")**

**print("Seleccione una opcion")**

***def* getComprobarNumero():**

**print("Introduzca un numero")**

**aux = input()**

**while (aux.isdigit() == False):**

**print("Esto no es un numero")**

**print("Introduzca un numero")**

**aux = input()**

**num = *int*(aux)**

**return num**

**#-----------------PRINCIPAL-------------------**

**print("Calculadora métodos")**

**numero1 = getComprobarNumero()**

**numero2 = getComprobarNumero()**

**opcion = 1**

**while (opcion != 0):**

**mostrarMenu()**

**opcion = *int*(input())**

**if (opcion == 1):**

**suma = sumarNumeros(numero1, numero2)**

**print(*f*"La suma de {numero1} y {numero2} es {suma}")**

**elif (opcion == 2):**

**multi = multiplicarNumeros(numero1, numero2)**

**print(*f*"{numero1} \* {numero2} = {multi}")**

**elif (opcion == 3):**

**numero1 = getComprobarNumero()**

**numero2 = getComprobarNumero()**

**print("Fin de programa")**

## LIBRERIAS EN PYTHON

Una librería es una funcionalidad con una serie de características de algún tipo.

Son métodos especializados en algo en particular, por ejemplo, en comprobar números cómo acabamos de hacer.

Dentro del lenguaje Python tenemos tres opciones:

1. Librerías integradas dentro del compilador de Python. Por ejemplo, actualmente, si deseamos leer algún fichero, no podemos, deberíamos llamar a una librería que contiene Python.

O si necesitamos saber la Tangente de un número o la potencia de un número existe una librería llamada **math**.

1. Librerías dentro de los Repositorios de **Pypi**. Para utilizar estas librerías es necesario tener el instalador de librerías **pip**

<https://pypi.org/>

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

1. Nuestras propias librerías creadas

Para utilizar librerías en nuestro programa tenemos varias formas de hacer las llamadas a las librerías:

1. Traer las librerías y acceder a sus métodos por su nombre

**PROGRAMA.PY**

import miLibreria

miLibreria.metodo1()

miLibreria.metodo2()

1. Traer la librería y acceder a sus métodos por su alias

import miLibreria as alias

alias.metodo1()

alias.metodo2()

1. Traer directamente los métodos necesarios y utilizarlos en nuestro programa como si fueran nativos

from miLibreria import metodo1, metodo2

metodo1()

metodo2()

Vamos a verlo con la librería **math**



Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Opción dos, que sería agregar directamente el método **pow** desde **math**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**Creación de nuestras propias librerías**

Las librerías propias están en ficheros separados, es decir, por un lado, tenemos un fichero Python de métodos y, por otro lado, la llamada a dichos métodos en una clase principal llamada **main.py**

**LIBRERÍA.PY**

def metodoAccion1():

def metodoReturn():

**MAIN.PY**

from librería.py import metodoAccion1, metodoReturn

Para esta nueva arquitectura que acabamos de aprender, cambiamos los nombres de nuestros programas.

* Librerías: **libraryXXdescripcion.py**
* Main: **mainXXdescripcion.py**

Para este ejemplo vamos a realizar lo siguiente:

* Tendremos una librería para las validaciones de un Email (lo tenemos) o de un NIF (Ahora lo hacemos).
* La librería tendrá dos métodos de validación y nos devolverá datos.

Para validar un NIF recibiremos un número y dividimos el número entre 23.

El resto de la división nos dará un número comprendido entre 0 y 23.

Cada número corresponde a una letra

Texto, Tabla

El contenido generado por IA puede ser incorrecto.

Creamos una nueva librería de validaciones llamada **library09validaciones.py**

**LIBRARY09VALIDACIONES**

***def* validarEmail(*email*):**

**if (email.find("@") == -1):**

**return False**

**elif (email.count(".") == 0):**

**return False**

**elif (email.startswith("@") == True or email.endswith("@") == True):**

**return False**

**elif (email.count("@") > 1):**

**return False**

**elif (email.find("@") > email.rfind(".")):**

**return False**

**else:**

**#posicion del ultimo punto**

**ultimoPunto = email.rfind(".")**

**dominio = email[ultimoPunto + 1:]**

**if (len(dominio) >= 2 and len(dominio) <= 3):**

**return True**

**else:**

**return False**

***def* getLetraNif(*numeroDni*):**

**resultado = numeroDni % 23**

**letrasDni="TRWAGMYFPDXBNJZSQVHLCKET"**

**letra = letrasDni[resultado]**

**return letra**

A continuación, creamos una clase principal que utilizará nuestra librería de métodos de validación.

Creamos una nueva clase llamada **main09validaciones.py**

**MAIN09VALIDACIONES.PY**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**import library09validaciones as li**

**print("Ejemplo librerías")**

**print("Introduzca un email")**

**email = input()**

**respuesta = li.validarEmail(email)**

**print(*f*"El email {email} es {respuesta}")**

**print("Introduzca su numero de NIF")**

**numero = *int*(input())**

**letra = li.getLetraNif(numero)**

**print(*f*"La letra del NIF {numero} es {letra}")**

**print("Fin de programa")**

## CONJUNTOS DE DATOS

Hasta ahora, hemos trabajado con datos simples, es decir, string, float, int o boolean.

Dentro de Python existen tipos de dato para almacenar múltiples objetos en su interior, un conjunto de números, textos o clases.

Todos los objetos conjuntos empiezan en CERO.

Tenemos tres tipos de objeto dentro de Python:

Siempre tendrá 3 elementos

1. **Listas**. Son elementos dinámicos que nos permiten almacenar múltiples valores en su interior.

Cada objeto almacenado se recupera mediante un índice dentro del conjunto

Las listas crecen o decrecen según trabajemos con su contenido.

Podemos almacenar en su interior cualquier elemento.

Para averiguar el tamaño de una lista se realiza mediante **len()**

Tenemos una serie de métodos para trabajar con listas:

* + **sort():** Ordena los elementos de una lista solo si es de tipo primitivo
  + **append(objeto):** Agrega un elemento al final de la lista
  + **insert(indice, objeto):** Inserta un elemento en una posición determinada. La posición dónde insertemos el elemento debe existir.
  + **remove(objeto):** Elimina el primer objeto que encuentra en la colección/lista
  + **pop(indice):** Elimina un objeto de la lista por su índice
  + **Recuperar un objeto de la lista:** lista[indice]

Para crear una lista en Python se utilizan los **CORCHETES[]**

Creamos un programa llamado **python10listas.py**

**print("Ejemplo de listas")**

**#La declaracion de una lista puede ser solo declaracion**

**#o agregar los objetos a su vez**

**listaNumeros = [12, 88, 14, 22, 16, 2]**

**#ORDENAR LA LISTA**

**#TODOS LOS METODOS MODIFICAN EL OBJETO**

**#ASCENDENTE**

**listaNumeros.sort()**

**listaNumeros.sort(*reverse*=True)**

**# for i in range(len(listaNumeros)):**

**#     print(listaNumeros[i])**

**#CREAMOS UNA LISTA CON NOMBRES**

**listaNombres = ["Adrian", "Lucia", "Carlos", "Antonia", "Diana", "Carlos"]**

**#LA LISTA CRECE SI AÑADIMOS DATOS**

**listaNombres.append("Sofia")**

**#PROBAMOS A INSERTAR UN NUEVO NOMBRE**

**#listaNombres.insert(11, "Infiltrado")**

**#Eliminamos un dato con Remove**

**# listaNombres.remove("Carlos")**

**#Eliminamos un dato que NO existe: Excepción**

**#listaNombres.remove("Carlitos")**

**#Probamos pop por indice**

**#listaNombres.pop(5)**

**#Si eliminamos una posicion pop que no existe?: Excepción**

**listaNombres.pop(15)**

**print(*f*"La longitud de nombres es {len(listaNombres)}")**

**#DIBUJAMOS LOS NOMBRES CON SU POSICION**

**for i in range(len(listaNombres)):**

**nombre = listaNombres[i]**

**print(*f*"{i} - {nombre}")**

**print("Fin de programa")**

**BUCLES DE REFERENCIA**

Cuando hablamos de conjutos, podemos recorrer los objetos del interior del conjunto de dos formas distintas.

La primera forma es la clásica que es un Bucle Contador

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Existe una forma de recorrer los objetos UNO a UNO utilizando cada objeto del interior del conjunto como parte del bucle, en lugar de utilizar i, se utiliza **name**

Perdemos la capacidad de tener un contador.

Solamente lo utilizamos para recorrer los objetos y acceder a ellos.

Se utiliza una variable de referencia, una variable de referencia apunta a un objeto que ya existe, no es un nuevo objeto.

La variable **nombre** hace referencia a una posición de un objeto dentro de la colección.

**nombre** y **listaNombres[i]** apuntan al mismo espacio de memoria

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Sintaxis:

for (VariableReferencia in CONJUNTO):

#instrucciones

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Un bucle de referencia es de SOLO LECTURA en su conjunto.

Texto

El contenido generado por IA puede ser incorrecto.

Si hacemos lo mismo con el bucle contador???

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**TUPLAS**

Una Tupla es como un Array, es estático, su tamaño no puede cambiar, pero se diferencia de un Array en que es de SOLO LECTURA.

Una vez que hemos almacenado los datos en su interior, no se puede modificar, es constante.

Una tupla se declara mediante **paréntesis ()**

Funciona igual que una lista, se utiliza **len()** o podemos acceder a cada dato con **tupla[indice]**

Creamos un nuevo programa llamado **python11tuplas.py**

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**DICCIONARIOS**

Un diccionario es un objeto conjunto con formato key:value, como un JSON.

En realidad, es un JSON, pero también puede ser tratado como un conjunto de Keys y Values

Las Keys NO pueden estar repetidas

Un diccionario es mutable, por lo que podemos agregar elementos en su interior

Es un JSON

{

“raza”: “Gato”, “edad”: 5, “nombre”: “Garfield”, “raza”: “Gato gordo”

}

Un diccionario, al ser creado se debe instanciar mediante la palabra clave **dict()**

diccionario = dict()

diccionario = {

“raza”: “Gato”, “edad”: 5, “nombre”: “Garfield”

}

Podemos recuperar valores del diccionario por su clave mediante **get()**

diccionario.get(raza) 🡪 Gato

Tenemos una forma de recorrer un diccionario con sus keys/values a la vez mediante **items()**

Sintaxis:

for (key, value in diccionario.**items()**):

print(key, value)

Por supuesto, como es un objeto conjunto, se utiliza **len()** para saber la longitud de claves que contiene.

**RECORRER SOLAMENTE LAS KEYS**

for (key in diccionario.**keys()**):

print(key)

**RECORRER SOLAMENTE LOS VALUES**

for (value in diccionario.**values()**):

print(value)

Podemos eliminar todo el objeto interior mediante **clear()**

También podemos eliminar una clave mediante **pop()**

Podemos agregar elementos mediante **setDefault()**

También podemos recuperar un elemento mediante objeto[INDICE] o mediante objeto[CLAVE]

Vamos a crear un simple ejemplo para manejar un diccionario cómo conjunto de claves y values

Creamos un nuevo programa llamado **python12diccionarios.py**

**print("Ejemplo de diccionarios")**

**#CREAMOS EL DICCIONARIO DE PROVINCIAS**

**provincias = *dict*()**

**provincias = {**

**924 : "Badajoz",**

**956:  "Cádiz",**

**958 : "Granada",**

**959 : "Huelva"**

**}**

**#AGREGAMOS UNA NUEVA PROVINCIA**

**provincias.setdefault(33, "Oviedo")**

**#QUE SUCEDE SI YA EXISTE¿¿?? NO LO INSERTA**

**provincias.setdefault(959, "Oviedo")**

**provincias.pop(959)**

**print(*f*"Longitud de Keys en diccionario: {len(provincias)}")**

**for key, value in provincias.items():**

**print(*f*"{key}: {value}")**

**print("Fin de programa")**

## CONTROL DE EXCEPCIONES

Un control de excepciones nos permite controlar el código cuando sucede algo imprevisto.

Cuando sucede un error, el programa se detiene y no continua. Esta lógica nos permite poder capturar la excepción y realizar acciones si lo necesitamos sin detener el programa.

El control de excepciones debería utilizarse en cualquier desarrollo al final y en los elementos que NO podamos controlar por código.

Por ejemplo, imaginemos que tenemos un programa que lee una base de datos, todo funciona siempre y es correcto. Pero pudiera ser que el servidor de BBDD no esté activo y no tenemos forma de controlar esa petición, ahí entraría el control de excepciones.

Los bloques de excepciones en Python se utilizan mediante **try…except**

Creamos un nuevo programa llamado **python13excepciones.py**

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Si introducimos una letra, veremos una excepción

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Las excepciones están delimitadas por un **TIPO**, que indica qué clase de excepción hemos tenido.

No será lo mismo abrir un fichero que convertir un número a texto.

Según vemos en la imagen, el tipo de excepción que estamos teniendo es **ValueError**

Para capturar esta excepción se utiliza la siguiente sintaxis:

**try:**

#CODIGO A EJECUTAR

**except NOMBRE\_EXCEPCION:**

#ENTRA SI LA EXCEPCION ES LANZADA

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

También tenemos la posibilidad de volver a pedir un número si nos lanza la excepción

Texto

El contenido generado por IA puede ser incorrecto.

Las excepciones van mediante tipos de excepción, también podemos capturar múltiples excepciones en múltiples bloques **except** y separa qué deseamos hacer con cada tipo de excepción.

Cada excepción será un **except**

Por ejemplo, vamos a modificar el código y dividiremos dos números.

Si el número divisor es cero, nos lanzará otra excepción distinta a ValueError.

Texto

El contenido generado por IA puede ser incorrecto.

La siguiente lógica es tener una excepción GENERAL, no importa el tipo de excepción, sino que yo quiero capturar todas.

Para esta sintaxis se utiliza **except** sin el tipo de excepción

Tenemos varias posibilidades:

1. Capturar varias excepciones y, si sucede un error que no he controlador, capturarlo también. **El bloque except siempre debe ser el último**

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

1. Capturar todas las excepciones a la vez, solamente tendremos un bloque **except**

Texto

El contenido generado por IA puede ser incorrecto.

Cuando capturamos una excepción general, existe una librería llamada **sys** con un método llamado **exc\_info()** que nos daría información de la excepción si deseamos mostrarla o hacer algo con ella.

Texto

El contenido generado por IA puede ser incorrecto.

Por último, por supuesto, yo nunca utilizaría estos ejemplos que os he puesto aquí, podemos comprobar perfectamente si un texto es un número o si el divisor es cero mediante IF.

Las excepciones solamente se utilizan cuando NO tengamos control de código.

Tenemos un último bloque que es bloque **finally**

Dicho bloque SIEMPRE se ejecuta, tanto si tenemos excepción como si no tenemos excepción.

Sintaxis:

**try:**

#CODIGO A EJECUTAR

**except NOMBRE\_EXCEPCION:**

#ENTRA SI LA EXCEPCION ES LANZADA

**finally:**

#CODIGO QUE SIEMPRE SE EJECUTA

El bloque finally se puede utilizar en el siguiente supuesto, imaginemos que tenemos un bloque se conecta y lee registros de una base de datos

Conexión.abrir()

Conexión.execute()

Conexión.read()

conexión.cerrar()

**try:**

Conexión.abrir()

Conexión.execute()

Conexión.read()

**except NOMBRE\_EXCEPCION:**

#ENTRA SI LA EXCEPCION ES LANZADA

**finally:**

#CODIGO QUE SIEMPRE SE EJECUTA

**conexión.cerrar()**

## PROGRAMACION ORIENTADA A OBJETOS

La programación orientada a objetos dentro de Python no contiene todas las características que existen algunos lenguajes más sólidos: Java, C#, pero si que tiene casi todas las bases.

Hemos creado un ejemplo sobre librerías, en realidad, una librería se podría denominar una clase de Python, pero más bien es una herramienta con métodos de Python, no es una clase aunque se puede denominar así.

Las clases lo que nos permiten es almacenar datos en el interior de un objeto y tratar dichos datos mediante métodos.

Una clase está compuesta por propiedades y métodos.

Por ejemplo, pongamos que necesitamos almacenar datos de una ciudad, nombre, temperatura o población.

Podemos tener tres variables separadas:

nombreCiudad = “Alicante”

temperatura = 35

población = 31000

Pongamos que necesitamos almacenar datos de múltiples ciudades:

listaNombresCiudad = [“Alicante”, “Gijon”, “Cartagena”]

listaTemperaturas = [35, 20, 39]

listaPoblaciones = [310000, 0000,0000]

En este supuesto, las clases cobran importancia, podríamos crear un objeto que contuviera todas las propiedades necesarias:

city= new Ciudad()

city.nombre = Alicante

city.temperatura = 35

city.poblacion = 31000

city.quemarHogueras()

La creación de una clase en Python se realiza con la siguiente sintaxis:

class NombreClase:

#PROPIEDADES Y METODOS

Dentro de la POO tenemos varios términos que definen dicho POO:

* **Encapsulación:** Solamente dejaremos ver lo que deseamos que sea visible.
* **Polimorfismo:** No soportado de forma nativa por Python. Indica que un método puede tener múltiples formas, por ejemplo un pedal de freno.
* **Herencia:** Una clase hereda todas las características de otra clase y las implementa.
* **Abstracción:** Podemos abstraernos y manejar varios objetos del mismo tipo siempre que no entremos en sus características propias del objeto.

**CONCEPTO DE CONSTRUCTOR**

Un constructor es un método de creación de un objeto.

Dicho método es el primero que se ejecuta cuando instanciamos una clase.

Nos permite realizar acciones al crear un objeto.

En **Python** un constructor (opcional) se define con la palabra clave **def** y con el nombre de método **\_\_init\_\_**

**PALABRAS CLAVES EN OBJETOS**

* **self:** Es obligatoria en los métodos de cualquier clase de Python

Nos permite acceder a los atributos/métodos de un objeto.

Debe ser el primer argumento de los métodos de una clase, si tiene más, irán a continuación.

* **super:** Este término se utiliza cuando estamos heredando de una clase. Indica la clase de la que estamos heredando para acceder a sus características.

Solamente se utiliza con la herencia.

Por ejemplo, si tenemos una clase Deportivo que hereda de un Coche.

Mediante **self** accedemos al objeto **Deportivo**

Mediante **super** accedemos al objeto **Coche**

Ejemplo de creación de una clase Lámpara:

class Lampara:

#PODEMOS DECLARA AQUÍ LAS PROPIEDADES

Encendida

Color

Brillo

def \_\_init\_\_(self):

self.Encendida = false

self.Color = Azul

self.Brillo = 40

def encender(self):

self.encendida = true

def cambiarBrillo(self, nuevoBrillo):

self.brillo = nuevoBrillo

Una persona utiliza la lampara:

miLampara = new Lampara()

#en este momento, cómo está la lámpara: Encendida/Apagada?

miLampara.encender()

miLampara.cambiarBrillo(80)

Las clases que tenemos actualmente están por ahí tiradas, tanto si son **mains** como si son **libraries**.

Deberíamos ser organizados con nuestras clases en nuestros proyectos.

Las clases se suelen denominar **models**

Asi que lo que haremos será tener una carpeta llamada **models** y no voy a llamar a los modelos XX.

Tendremos una clase **python14pruebapersonas.py** que será la que utilizaremos para visualizar lo que estamos haciendo

Creamos una carpeta llamada **models** y una clase llamada **persona.py**

**PERSONA**

***class* Persona:**

**#TENEMOS DOS FORMAS DE DECLARAR PROPIEDADES**

**#PARA LA PERSONA**

**#1) AL INICIO DE LA CLASE**

**# nombre = ""**

**# apellidos = ""**

**# edad = 0**

**# pais = ""**

***def* \_\_init\_\_(*self*):**

**#tenemos la posibilidad de crear propiedades privadas**

**#las propiedades privadas solamente se pueden utilizar dentro**

**#de la clase Persona y se definen mediante dos guiones bajos:**

**#\_\_propiedadPrivada**

***self*.*\_\_datoPrivado* = "Esto no lo vas a ver Python..."**

***self*.*nombre* = ""**

***self*.*apellidos* = ""**

***self*.*edad* = 0**

***self*.*pais* = "Francia"**

***def* getDatoPrivado(*self*):**

**return *self*.*\_\_datoPrivado***

**#CREAMOS UN METODO QUE NOS DEVUELVA EL NOMBRE COMPLETO DE UNA PERSONA**

***def* getNombreCompleto(*self*):**

**return *self*.*nombre* + " " + *self*.*apellidos***

**#imaginemos que tenemos un método que deseamos que nos**

**#devuelva el nombre completo, pero deseamos que lo realice**

**#al reves, apellidos y nombre**

**# def getNombreCompleto(self, reves):**

**#     return self.apellidos + " " + self.nombre**

Python no permite sobrecarga/polimorfismo por defecto, necesitaríamos una librería externa de **pip** que nos ayude

Texto

El contenido generado por IA puede ser incorrecto.

**PYTHON14PRUEBASPERSONAS**

**from models.persona import Persona**

**print("Utilizando clases Persona")**

**personaje = Persona()**

**print(personaje.*pais*)**

**personaje.*nombre* = "Sheldon"**

**personaje.*apellidos* = "Cooper"**

**personaje.*edad* = 44**

**personaje.*pais* = "EEUU"**

**print(personaje.getDatoPrivado())**

**print(personaje.getNombreCompleto(22))**

**# print(personaje.getNombreCompletoReves())**

**print(*f*"Nombre: {personaje.*nombre*}, Apellidos: {personaje.*apellidos*}")**

## HERENCIA DE CLASES

La herencia implica que una clase recupera todas las características de otra clase y las mejora.

Vamos a crear una nueva clase llamada **empleado.py** dentro de **models** y heredaremos de **Persona**

Para realizar la herencia dentro de Python se hace con la siguiente sintaxis:

class ClaseHereda(**ClaseSuper**):

Texto

El contenido generado por IA puede ser incorrecto.

Cuando realizamos herencia, se recuperan las propiedades pero no son iniciadas, es decir, cuando creamos un empleado NO está leyendo el constructor de PERSONA.

Si deseamos (buena praxis) debemos llamar al constructor de **Persona** de forma explicita

Para llamar al constructor o lo que sea de la clase que estamos heredando se utiliza la palabra clave **super**

Texto

El contenido generado por IA puede ser incorrecto.

**SOBRESCRIBIR METODOS**

Cuando hablamos de herencia tenemos dos características:

* **Implementación**: Crear métodos o propiedades que no existan en la clase super

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

* **Sobrescritura**: La sobrescritura de un método implica adaptar un método de la clase SUPER a nuestra propia clase.

Para sobrescribir necesitamos que el método que hereda se llame igual que el método de la clase **super()**

Por ejemplo, tenemos un **Empleado** que tiene 22 días de vacaciones.

Quiero crear un **Director** que también tenga vacaciones y que tenga **30** días.

Actualmente, los dos tienen los mismos días de vacaciones

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Si sobrescribimos el método en **Director**

Texto

El contenido generado por IA puede ser incorrecto.

Ya tenemos lo que deseamos

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

En el concepto de la sobrescritura tenemos dos opciones:

1. Método aislado, es decir, no tiene nada que tenga que ver con el método que hereda
2. Método Reutilizado: Necesito lógica del método de la clase **super()** y utilizar dicha lógica en mi clase

Si ahora modificamos nuestro supuesto:

Por ejemplo, tenemos un **Empleado** que tiene 22 días de vacaciones.

Quiero crear un **Director** que también tenga vacaciones y que tenga **8 días más** que un empleado, es decir, **30** días actualmente.

Texto

El contenido generado por IA puede ser incorrecto.

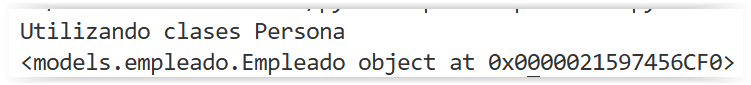
Con el mismo concepto que tenemos actualmente, existen métodos nativos que nos permiten adaptar nuestras clases a nuestras necesidades.

Por ejemplo, tenemos un método **\_\_str\_\_** que representa un objeto en su formato String cuando lo dibujamos por pantalla.

Pongamos que dibujamos un objeto Empleado.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.



Si sobrescribimos el método **\_\_str\_\_** podemos adaptar el dibujo que deseemos cuando alguien invoca al método ToString de un objeto

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.