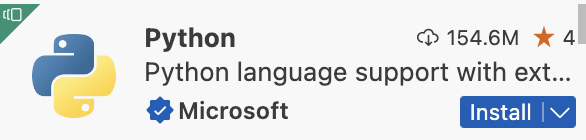
Lo primero de todo es descargar el compilador de Python en nuestro equipo

<https://www.python.org/>

El siguiente paso es descargar y configurar nuestro IDE.

Descargamos Visual Studio Code

Debemos instalar extensiones dentro de Visual Studio

Vamos a comprobar si lo ha realizado correctamente

Tendremos una carpeta llamada Python y desde ahí incluiremos nuestros ficheros de Python que finalizan con .py

Cuando creamos ficheros de programación debemos tener en cuenta algunas características

1. Sin espacios en los nombres
2. Sin caracteres especiales
3. No deben comenzar con número

Nosotros vamos a utilizar una nomenclatura visual para saber los ficheros de Python que realizamos

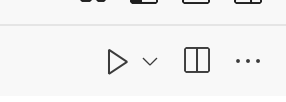
python01test.py

El lenguaje diferencia mayúsculas de minúsculas

Mostramos un mensaje en nuestro programa

Los textos que escribámosles de forma literal deben ir entre comillas dobles

“Hoy es juernes”

Para mostrar un mensaje se utiliza la palabra

print(“Nuestro mensaje”)

Para ejecutar el código, pulsamos sobre el Play

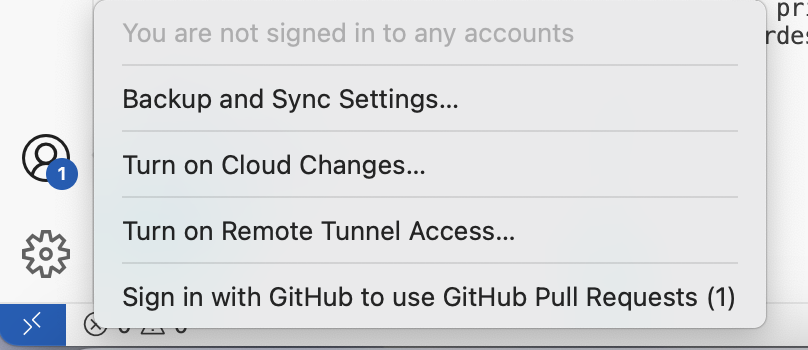
Ya podremos visualizar nuestro primer programa

INSTALAR GIT Y CONFIGURARLO

Windows:

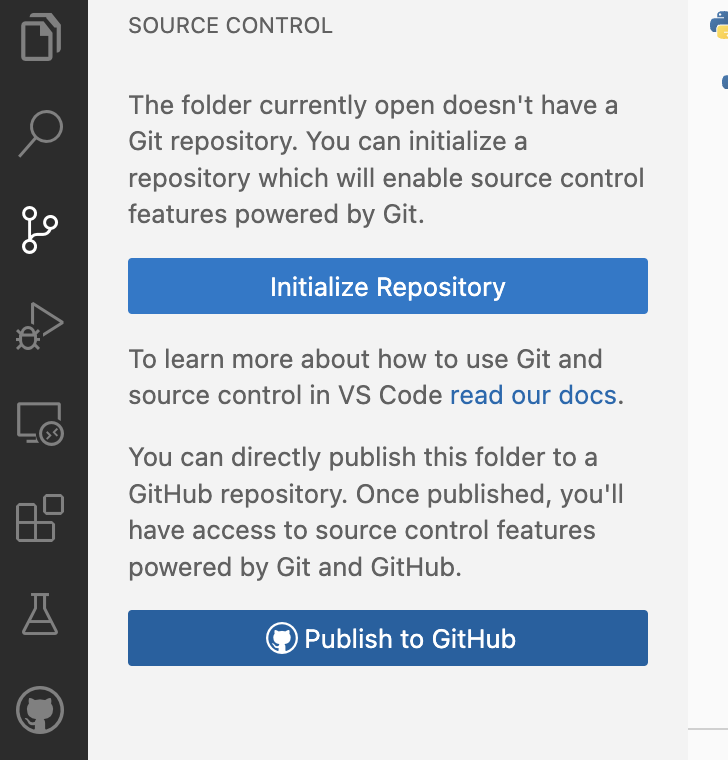
<https://git-scm.com/downloads/win>

Agregamos la extensión de Git Hub dentro de Visual Studio Code

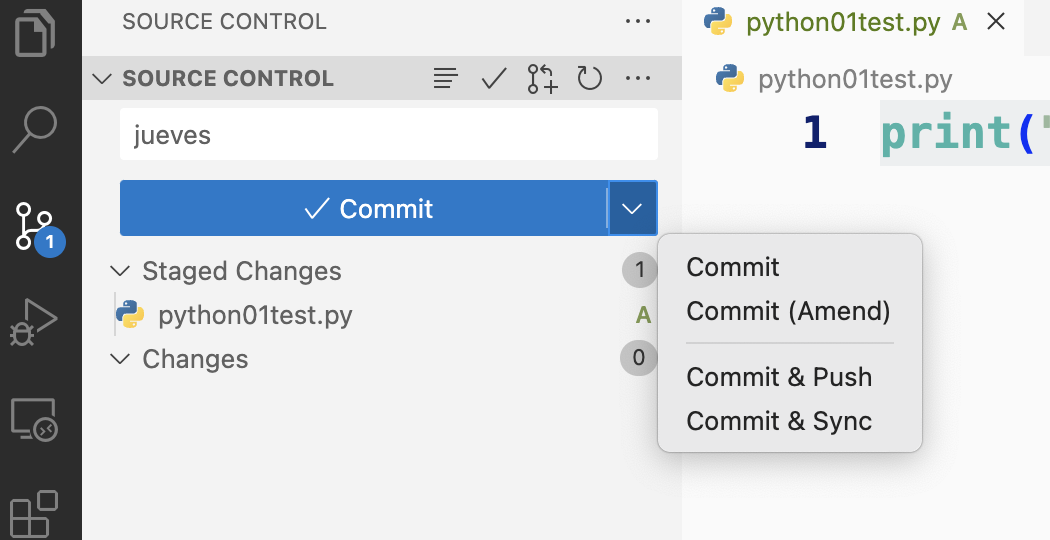
Nos validamos con GitHub

Una vez que nos hemos validado tendremos una opción en el menú que será Backup and Sync Settings

Esto nos permite sincronizar nuestras extensiones y personalizaciones de Visual Studio Code

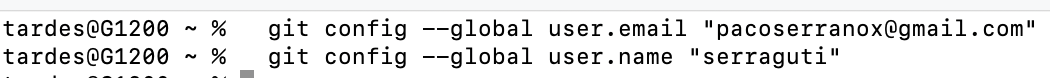
A continuación vamos a subir nuestro Repositorio a GitHub. Debemos pulsar en Publish to Github

Al intentar subir a la nube nuestros ficheros, nos indica que no puede realizarlo porque no nos hemos dado de alta en el sistema operativo

Debemos abrir una ventana de comandos ( cmd ) y escribir las siguientes instrucciones

**git config --global user.email "you@example.com"**

**git config --global user.name "Your usuario github "**



Vamos a visualizar cómo podemos descargar nuestro proyecto y utilizarlo dentro de Git.

1.- Creamos una nueva carpeta dentro de Documento llamada pruebagit

2.- Abrimos esa carpeta con VS Code y creamos un nuevo fichero Python llamado test.py

print(“Soy una prueba”)

3.- Subimos nuestra carpeta a Github en un nuevo repositorio

4.- Hacemos cambios en el fichero test.py y los subimos a Github

5.- Cerramos visual studio y eliminamos la carpeta dentro de Documents llamada pruebagit

6.- Sincronizamos de nuevo nuestra carpeta con Git (descargar con Git Clone)

PYTHON

Es un lenguaje de programación, nos permite, mediante instrucciones poder ordenar características a nuestros equipos/programas

Todos los lenguajes tienen unas normas:

Los ficheros deben terminar en .py

No deben comenzar con número ni tener caracteres extraños

Todo el código (por ahora) estará dentro de un mismo archivo.

Muy importante: Las tabulaciones en Python son básicas, delimitan el código del programa

Las variables no se declaran con tipado, se asigna un valor y ese será su tipo de dato.

Numero = 2

Texto = “Mi string”

Si las variables son números, podremos sumar, restar o dividir

Si son textos, podremos convertir a mayúsculas o recorrer letra a letra

Creamos un nuevo fichero python02variables.py

**print("Ejemplo variables")**

**#ESTO ES UN COMENTARIO**

**#COMENTAR: COMMAND + K + C**

**# DESCOMENTAR: COMMAND + K + U**

**# Las variables no se declaran con tipado**

**# las variables las declaramos en minuscula**

**numero = 99**

**texto = "Variable de tipo texto (string)"**

**print(numero)**

**print(texto)**

Operaciones matemáticas

* Suma
* Resta

/ división

* Multiplicación

% Resto

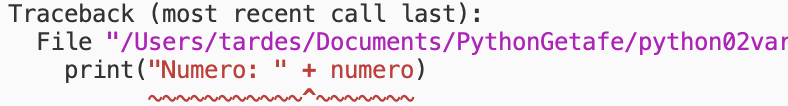
\*\* POTENCIA

Para poder dibujar utilizamos print, pero podemos agregar algo más a print, es decir, lo que se llama una concatenación.

Mostrar un texto junto a una variable

print(“TEXT”, variable)

Tenemos otra posibilidad de concatenar que es utilizar el símbolo universal (+), esto nos permite realizar mensajes más complejos

Veremos que nos está dando un error:

Si utilizamos un + con una variable INT, está intentado sumar un string con un número.

Para solucionar esto, tenemos una funcionalidad llamada Funciones de conversión.

Python se basa en funciones. Una función es un bloque de código que “hace algo”.

Funciones de conversión:

\* str(variable): Convierte una variable a tipo string

* float(variable): Convierte una variable a decimal

\* int(variable): Convierte una variable a número entero

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

Las funciones de conversión NO hacen milagros.

Dentro de cualquier programa tenemos dos tipos de errores:

* Errores de compilación: Algo está mal escrito en el programa y el programa no compila
* Errores en ejecución: El programa funciona y compila correctamente, pero tiene algún fallo en algún momento de la ejecución
* Errores lógicos: Todo funciona, mi programa va perfecto, no tengo ningún fallo, pero no hace lo que quiero.

Creamos un nuevo fichero llamado **python03mates.py**

**print("Ejemplo mates variables")**

**numero1 = 20**

**numero2 = 3**

**suma = numero1 + numero2**

**multiplicacion = numero1 \* numero2**

**division = numero1 / numero2**

**resta = numero1 - numero2**

**print("Resta ", resta)**

**print(str(numero1) + " \* " + str(numero2) + "=" + str(multiplicacion))**

**print("División ", division)**

**print("Suma: ", suma)**

**redondeo = int(division)**

**print("Redondeo division: ", redondeo)**

PEDIR VALORES AL USUARIO

Hemos utilizado valores estáticos, pero a continuación vamos a combinar la petición de valores para las variables dentro del programa.

Para pedir valores se utiliza la función input()

Cuando pedimos valores a un usuario, siempre serán de tipo STRING, aunque pidamos valores numéricos

Modificamos el ejemplo anterior y pedimos valores al usuario

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

CONDICIONALES

Un condicional es una pregunta dentro del código

Dicho código ya no será lineal, sino que irá por un lado o por otro, dependiendo de las preguntas.

Los condicionales se escriben mediante la palabra if

Para utilizar condicionales dentro de Python debemos aplicar TABULACIONES

Mediante las tabulaciones, será código del IF o código del resto del programa

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Necesitamos saber los operadores de comparación

> Mayor

>= Mayor o igual

< Menor

<= Menor o igual

== Igual

**!=**

Distinto

Creamos un nuevo programa llamado python04condicionales.py

Texto

El contenido generado por IA puede ser incorrecto.

También tenemos la posibilidad de utilizar un ELSE, en el caso de que la condición no se cumpla

Texto

El contenido generado por IA puede ser incorrecto.

Creamos un nuevo fichero llamado python05positivo.py

Pedimos un número al usuario e indicamos si es Positivo o negativo o cero

**print("Número positivo/negativo/cero")**

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

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

**if (numero > 0):**

**print("Positivo")**

**else:**

**if (numero == 0):**

**print("Cero")**

**else:**

**print("Negativo")**

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

Tenemos una sintaxis que nos permite preguntar de nuevo sobre una misma variable dentro de un IF, se llama ELSE IF y en Python se escribe elif

Modificamos nuestro código anterior para mostrar la nueva sintaxis:

**print("Número positivo/negativo/cero")**

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

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

**if (numero > 0):**

**print("Positivo")**

**elif (numero == 0):**

**print("Cero")**

**else:**

**print("Negativo")**

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

Vamos a crear otro fichero llamado **python06numeromayor.py**

Pedimos al usuario dos números y mostramos qué número es mayor o mostramos si los números son iguales

**print("Número mayor dos números")**

**print("Introduzca número 1")**

**numero1 = *int*(input())**

**print("Introduzca número 2")**

**numero2 = *int*(input())**

**if (numero1 > numero2):**

**print("El número " + *str*(numero1) + " es mayor que " + *str*(numero2))**

**elif (numero1 == numero2):**

**print("Los dos números son iguales")**

**else:**

**print("El número mayor es ", numero2)**

**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 y que no están por defecto dentro de nuestro programa.

Tenemos varios tipos de librerías, Nativas de Python y otras que tendremos que instalar de forma externa.

Por ejemplo, si deseamos leer un fichero, ahora mismo NO podemos, nuestro programa no tiene las herramientas. Debemos darle una librería para que pueda leer.

Por ejemplo, si deseamos leer una base de datos Oracle, el programa no tiene ni idea, no tiene ni las herramientas ni las librerías.

Tenemos una librería que nos permite saber la tangente o el coseno con operaciones.

Dichas operaciones si deseamos utilizarlas en nuestro programa debemos **importar la librería math**

**PROGRAMA PY**

import miLibreria

miLibreria.Metodo1()

miLibreria.Metodo2()

Tenemos otra sintaxis:

**PROGRAMA PY**

from miLibreria import Metodo1, Metodo2

Metodo1()

Metodo2()

Vamos a visualizar un ejemplo sencillo con **math**

Creamos un nuevo fichero llamado **python07libreriamath.py**

**print("Ejemplo de librerías")**

**#SINTAXIS CON from**

**from math import floor, ceil, trunc**

**numero1 = 20**

**numero2 = 3**

**division = numero1 / numero2**

**print("La división es ", division)**

**# DECLARAMOS VARIABLES PARA ALMACENAR LOS VALORES**

**varFloor = floor(division)**

**varCeil = ceil(division)**

**varTrunc = trunc(division)**

**print("Floor ", varFloor)**

**print("Ceil ", varCeil)**

**print("Trunc ", varTrunc)**

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

**DEBUG APLICACIONES**

La depuración de aplicaciones es algo básico dentro de cualquier entorno de desarrollo.

Esta funcionalidad nos permite visualizar que va sucediendo con nuestro código

Paso a paso, podemos ir por los diferentes elementos de nuestro código y ver que sucede.

La depuración se utiliza para errores lógicos.

Vamos a visualizar la funcionalidad con el programa de math.

Debug en VS Code

1. Debemos indicar la línea donde desamos comenzar la depuración en nuestro código. Se puede realizar mediante la tecla F9

Texto

El contenido generado por IA puede ser incorrecto.

1. Pulsamos sobre **Run and Debug** en el menú lateral

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

El contenido generado por IA puede ser incorrecto.

Una vez ejecutado, nuestro programa se detendrá en esa línea seleccionada

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En la parte de la izquierda podremos visualizar lo que está haciendo nuestra aplicación con el valor de las variables

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

El contenido generado por IA puede ser incorrecto.

OPERADORES RELACIONALES

Estos operadores nos permiten realizar más de una pregunta dentro del mismo IF

Tenemos dos palabras para realizar estas consultas:

* **and: Todas las condiciones deben cumplirse**
* **or: Cualquier condición entrará en el IF**

Vamos a realizar un programa en el que pediremos tres números.

El programa nos dirá el mayor, el menor y el intermedio.

Creamos un nuevo programa llamado **python10mayortresnumeros.py**

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

SOLUCION

**print("Ejemplo mayor tres numeros")**

**print("Introduzca numero 1")**

**numero1 = *int*(input())**

**print("Introduzca numero 2")**

**numero2 = *int*(input())**

**print("Introduzca numero 3")**

**numero3 = *int*(input())**

**mayor = 0**

**menor = 0**

**intermedio = 0**

**#COMPARAMOS CADA NUMERO CON LOS OTROS DOS**

**if (numero1 >= numero2 and numero1 >= numero3):**

**mayor = numero1**

**elif (numero2 >= numero1 and numero2 >= numero3):**

**mayor = numero2**

**else:**

**mayor = numero3**

**#MISMA PREGUNTA PERO CAMBIANDO A SIMBOLO MENOR**

**if (numero1 <= numero2 and numero1 <= numero3):**

**menor = numero1**

**elif (numero2 <= numero1 and numero2 <= numero3):**

**menor = numero2**

**else:**

**menor = numero3**

**suma = numero1 + numero2 + numero3**

**intermedio = suma - mayor - menor**

**print("Mayor ", mayor)**

**print("Menor ", menor)**

**print("Intemedio ", intermedio)**

Creamos un nuevo programa **python11horasextra.py**

CALCULAR SALARIO DE TRABAJADORES

* Pediremos a un usuario las horas trabajadas, precio hora y los kilómetros

El trabajador tendrá horas extra a partir de la hora 36

Cada hora extra será 2€ más

Solamente mostraremos mensajes

* Si el trabajador hace menos de 100 km las dietas son LOCALES
* Si el trabajador hace entre 101km y 500 km las dietas son PROVINCIALES
* Si hace más km, serán NACIONALES
* Si el precio final es menor a 250€ SIN RETENCIONES
* Si es entre 250 y 600, 20% retención
* Si es mayor 40 % retención

**Número horas: 20**

Horas extra: 4

**Precio hora: 20**

Salario base: 4000

Salario Extra: 0

Salario total: 4000

Texto

El contenido generado por IA puede ser incorrecto.

SOLUCION

**print("Ejemplo Horas extra")**

**print("Horas trabajadas")**

**horasTrabajadas = *int*(input())**

**print("Precio hora")**

**precioHora = *int*(input())**

**print("Kilometros recorridos")**

**km = *int*(input())**

**horasExtra = 0**

**salarioExtra = 0**

**salarioBase = 0**

**salarioTotal = 0**

**dietas = ""**

**retencion = ""**

**#PREGUNTAMOS SI TENEMOS HORAS EXTRA**

**if (horasTrabajadas > 36):**

**#HORAS EXTRA**

**horasExtra = horasTrabajadas - 36**

**salarioBase = precioHora \* 36**

**salarioExtra = horasExtra \* (precioHora + 2)**

**else:**

**#NO HA HECHO HORAS EXTRA**

**horasExtra = 0**

**salarioExtra = 0**

**salarioBase = horasTrabajadas \* precioHora**

**salarioTotal = salarioBase + salarioExtra**

**if (km <= 100):**

**dietas = "LOCALES"**

**elif (km >= 101 and km <= 500):**

**dietas = "PROVINCIALES"**

**else:**

**dietas= "NACIONALES"**

**if (salarioTotal < 250):**

**retencion = "SIN RETENCION"**

**elif (salarioTotal >= 250 and salarioTotal <= 600):**

**retencion = "20% Retencion"**

**else:**

**retencion = "40% Retención"**

**#INFORME**

**print("Informe de salario")**

**print("Horas trabajadas ", horasTrabajadas)**

**print("Horas extra ", horasExtra)**

**print("Precio hora ", precioHora)**

**print("Precio extra ", (precioHora + 2))**

**print("Salario base ", salarioBase)**

**print("Salario extra ", salarioExtra)**

**print("Salario total ", salarioTotal)**

**print("Retenciones ", retencion)**

**print("Dietas ", dietas)**

BUCLES

Un bucle es una secuencia que se repite con una condición o una serie de veces.

Tenemos dos tipos de bucles:

1. **while:** Se realiza mediante una condición. Mientras que la condición no se cumpla, permanece dentro del bucle.

Debemos indicar, de alguna forma dentro del bucle, que la condición se debe cumplir.

Tenemos la posibilidad de realizar bucles infinitos

Sintaxis del bucle while:

while (condicion == true):

# instrucciones

condicion = false

1. **for:** Este bucle nos permite repetir una secuencia N veces. La secuencia se realiza con un contador declarado en la sintaxis del propio bucle.

Entra y sale solo, no hace bucles infinitos

Sintaxis Bucle Contador (for)

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

#instrucciones

La variable **contador** siempre comienza en CERO con esta sintaxis

Tenemos una sintaxis para indicar el inicio y fin del bucle

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

#instrucciones

Creamos un nuevo programa llamado **python12bucles.py**

**print("Ejemplo bucles")**

**print("WHILE")**

**# NECESITAMOS UNA VARIABLE PARA LA CONDICION DEL BUCLE**

**contador = 0**

**while (contador <= 5):**

**# DEBEMOS INDICAR QUE SALDREMOS DEL BUCLE**

**print("Contador ", contador)**

**contador = contador + 1**

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

Tabla

El contenido generado por IA puede ser incorrecto.

**print("Ejemplo bucles")**

**print("For")**

**# NORMALMENTE LAS VARIABLES DE LOS BUCLES CONTADORES**

**# SE REPRESENTAN CON UNA SOLA LETRA (i, z, j)**

**for i in range(5):**

**print("Valor de i ", i)**

**for i in range(1 , 6):**

**print("Valor i: ", i)**

**print("WHILE")**

**# NECESITAMOS UNA VARIABLE PARA LA CONDICION DEL BUCLE**

**contador = 0**

**while (contador <= 5):**

**# DEBEMOS INDICAR QUE SALDREMOS DEL BUCLE**

**print("Contador ", contador)**

**contador = contador + 1**

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

Vamos a realizar una aplicación en la que pediremos al usuario un número inicial y un número final y mostraremos los números pares que existan entre los dos.

Creamos un nuevo programa llamado **python13rangopares.py**

**print("Rango de números pares")**

**print("Introduzca un inicio")**

**inicio = *int*(input())**

**print("Introduzca un valor final")**

**fin = *int*(input())**

**# REALIZAMOS UN BUCLE DESDE UN INICIO HASTA UN FINAL + 1**

**for i in range(inicio, fin + 1):**

**# PREGUNTAMOS SI EL NUMERO ES PAR**

**if (i % 2 == 0):**

**print(i)**

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

Vamos a realizar un ejemplo para demostrar la conjetura de Collatz

La conjetura dice lo siguiente:

Todo número positivo siempre llegará a ser 1 cumpliendo estas dos condiciones:

* Si el número es PAR, dividimos entre 2
* Si el número es IMPAR, multiplicamos \* 3 y sumamos 1

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

Creamos un nuevo programa llamado **python14collatz.py**

**print("Conjetura de Collatz")**

**print("Introduzca un número")**

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

**while (numero != 1):**

**if (numero % 2 == 0):**

**numero = *int*(numero / 2)**

**else:**

**numero = numero \* 3 + 1**

**print(numero)**

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

Realizamos un programa nuevo en el que mostraremos la tabla de multiplicar de un número.

Pediremos al usuario el número y le mostraremos la siguiente tabla

5 \* 1 = 5

5 \* 2 = 10

Texto

El contenido generado por IA puede ser incorrecto.

**print("Tabla multiplicar")**

**print("Introduzca número")**

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

**for i in range(1, 11):**

**operacion = numero \* i**

**print(*str*(numero) + " \* " + *str*(i) + "=" + *str*(operacion))**

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

**CLASE STRING PYTHON**

Las clases, por pertenecer a algún tipo (string) tienen una serie de métodos para trabajar con dichos tipos.

Existen clases simples como, por ejemplo, la clase int que no tiene nada o clases más complejas como pueden ser la clase string o una clase que acceda a leer ficheros.

Una clase de objeto contiene una serie de métodos que tienen que ver con su tipado, por ejemplo, dentro de la clase string tenemos un método que nos permite poner el texto en mayúsculas.

Tenemos un método para saber la longitud de un texto: **len()**

Todo texto comienza siempre en índice CERO.

En realidad un string es un conjunto de caracteres, podemos recorrer cada carácter si lo deseamos.

Se diferencia mayúsculas de minúsculas en los string

texto = “Lenguaje Python”

longitudTexto = len(texto) 🡪 15

También tenemos la posibilidad de acceder a cada carácter mediante su posición de índice

Objeto[**índice**]

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 con string tenemos la posibilidad de extraer subcadenas de un texto.

En Python a esto se le llama **STRING SLICING**

texto = “primer python”

Tenemos dos posibilidades para extraer subcadenas:

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

texto[8:] 🡪 YTHON

1. Devuelve desde una posición inicial hasta una posición final. objeto[inicio:final]

texto[0:2] 🡪 PR

Creamos un nuevo programa **python16string.py**

**print("Clase string y funciones")**

**texto = "primero python"**

**#VAMOS PROBANDO METODOS Y VIENDO QUE DEVUELVEN**

**print("upper ", texto.upper())**

**print("replace " + texto.replace("o", "@"))**

**print("Letra 0: " + texto[0])**

**print("Logitud (len)", len(texto))**

**print("find P: ", texto.find("p"))**

**print("find Z: ", texto.find("z"))**

**# SOBRECARGA DE FIND (contenido a buscar, indice)**

**print("find p sobrecarga ", texto.find("p", 1))**

**print("rfind p ", texto.rfind("p"))**

**print("startswith A ", texto.startswith("A"))**

**print("endswith n ", texto.endswith("n"))**

**print("isdigit() ", texto.isdigit())**

**print("isalpha() ", texto.isalpha())**

**print("isalnum() ", texto.isalnum())**

**# Vamos a visualizar que pasa con SLICING**

**# SUBSTRING**

**# QUEREMOS RECUPERAR DESDE LA POSICION 2 EN ADELANTE**

**substring = texto[2:]**

**print("Posición 2 en adelante ", substring)**

**# EN PYTHON PODEMOS RECUPERAR UNA SUBCADENA**

**# DESDE UNA POSICION (2) A OTRA POSICION (5)**

**subtexto = texto[2: 5]**

**print("texto[2: 5] ", subtexto)**

**# PODEMOS RECORRER CADA CARACTER DE UN TEXTO**

**longitud = len(texto)**

**for i in range(longitud):**

**letra = texto[i]**

**print("Posición " + *str*(i) + ": " + letra)**

**# PODEMOS HASTA COMPROBAR QUE EL USUARIO HA ESCRITO NUMEROS O NO**

**print("Introduzca un número")**

**# PRIMERO A UNA VARIABLE AUXILIAR**

**aux = input()**

**if (aux.isdigit()):**

**print("Esto es un número!!!")**

**else:**

**print("No me has dado un número, campeón...")**

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

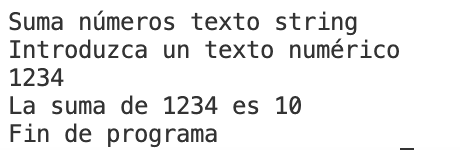
Necesito un programa llamado **python17sumanumeros.py**

Pediremos al usuario un texto numérico

“1234”

Debemos sumar cada carácter del texto y mostrar la suma de cada número/letra

**La suma es 10**



**print("Suma números texto string")**

**print("Introduzca un texto numérico")**

**textoNumeros = input()**

**suma = 0**

**longitud = len(textoNumeros)**

**# RECORREMOS CADA LETRA DEL TEXTO**

**for i in range(longitud):**

**# ALMACENAMOS CADA LETRA DE CADA POSICION**

**letra = textoNumeros[i]**

**# CONVERTIMOS CADA LETRA A INTEGER**

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

**# SUMAMOS CADA NUMERO**

**suma = suma + numero**

**print("La suma de " + textoNumeros + " es " + *str*(suma))**

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

Vamos a realizar un programa para validar un Email con Python

El usuario introducirá un email y le indicaremos si está mal y la razón.

**NO QUIERO BUCLES, TODO CON METODOS**

* Que el email contenga @
* Que el email tenga un punto
* @ ni al principio ni al final
* Punto ni al principio ni al final
* Que el email solamente tenga una @
* Que exista un punto después de la @
* El dominio debe ser de 2 a 3 caracteres

SOLUCION

**print("Validacion Email")**

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

**email = input()**

**# EMAIL CON @**

**if (email.count("@") == 0):**

**print("Email sin @")**

**# QUE EXISTA UN PUNTO**

**elif (email.find(".") == -1):**

**print("Email sin punto")**

**# EMAIL SIN @ AL INICIO NI FINAL**

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

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

**# Punto ni al inicio ni al final**

**elif (email[0] == "." or email.endswith(".") == True):**

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

**# Email con una sola @**

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

**print("Existe más de una @")**

**# Exista un punto despues de @**

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

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

**else:**

**# Necesitamos recuperar el dominio**

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

**# RECUPERAMOS EL DOMINIO A PARTIR DEL ULTIMO PUNTO EN ADELANTE**

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

**# LONGITUD DEL DOMINIO**

**longitudDominio = len(dominio)**

**# Comprobar la longitud 2-3**

**if (longitudDominio >= 2 and longitudDominio <= 3):**

**print("Email correcto " + email)**

**else:**

**print("El email debe tener un dominio de 2 a 3 caracteres")**

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

## FUNCIONES/METODOS DENTRO DEL LENGUAJE PYTHON

Una función es un método.

Son herramientas que contienen los programas de Python y las clases de Python.

Clase de Python: (string) upper()

Programas de Python: python19email: len(dominio)

Un programa de Python se ejecuta y UTILIZA clases de Python

Los métodos/funciones nos permiten realizar códigos NO LINEALES

Actualmente mi programa realiza la validación de un Email.

¿Qué sucede si también queremos validar un ISBN?

¿Qué sucede si también queremos validar un DNI?

Un método nos permite poder reutilizar código además de organizar nuestros programas.

Ya no estará el código lineal, sino que haremos las llamadas a otros códigos cuando lo necesitemos

Funciona mediante llamadas y declaraciones de métodos

PROGRAM PYTHON (main)

print(“Validacion Email”)

print(“Introduzca un email”)

email=input()

**validarEmail()**

En otro lugar del código o programa Python:

validarEmail(emailRecibido):

# REALIZAMOS LAS VALIDACIONES

Para crear métodos dentro de Python se utiliza una palabra clave llamada **def** y a continuación el nombre del método

Los métodos pueden estar en el mismo fichero Program main o en otros ficheros.

Si están en el mismo Program, lo primero que debemos hacer es escribir los métodos que necesitaremos.

Los métodos/funciones pueden recibir parámetros

Creamos un nuevo programa llamado **python20metodos.py**

**# ES UN PROGRAMA QUE DESEAMOS EJECUTAR**

**# LOS METODOS SE DECLARAN AL INICIO**

**# LA SINTAXIS DE NOMBRES DE METODOS ES**

**# primeraSegundaTercera**

***def* primerMetodo():**

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

**# LLAMAMOS DE FORMA EXPLICITA**

**print("Soy el primer metodo")**

***def* segundoMetodo():**

**print("Segundo metodo")**

**# AQUI TENEMOS EL CODIGO PRINCIPAL**

**print("Ejemplo de metodos")**

**# PODEMOS LLAMAR AL METODO SI LO DESEAMOS**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

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

### PARAMETROS EN LOS METODOS

Un método puede recibir parámetros, es decir, información para la ejecución del método.

Los parámetros se reciben dentro de los paréntesis del método y, si recibimos más de un parámetro, irán separados mediante comas.

Los nombres de parámetro no tienen que coincidir con los nombres de variable que estamos enviando.

Los parámetros solamente se utilizarán dentro del método.

Ejemplo:

def metodoParametros(**nombre**):

print(“Su nombre es “ + nombre)

name = input()

metodoParametros(**name**)

def metodoParametros(**nombre, num1, num2**):

Creamos un nuevo programa llamado **python21metodosparametros.py**

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

**print("Bienvenido a Python Mr/Mrs " + nombre)**

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

**print("Un placer hoy " + dia + " Mr/Mrs " + nombre)**

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

**print("Metodos con parametros")**

**name = "Alumno"**

**dia = "miercoles"**

**saludar(name)**

**despedirse("Jueves", dia)**

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

Los métodos podemos dividirlos entre dos grupos:

* Métodos de acción: Ejecutan una serie de acciones
* Métodos **return**: Ejecutan una serie de acciones y devuelven un valor después de ejecutarlas

Acción: **input()**

Return: len()

Creamos un nuevo programa llamado **python22metodosreturn.py**

***def* convertirMayusculas(*texto*):**

**return texto.uppper()**

***def* convertirMinusculas(*texto*):**

**return texto.lower()**

***def* concatenar(*texto1*, *texto2*):**

**resultado = texto1 + texto2**

**return resultado**

***def* mostrarMenu():**

**print("Seleccione una opción")**

**print("1.- Convertir mayúsculas")**

**print("2.- Convertir minúsculas")**

**print("3.- Concatenar textos")**

**# PROGRAMA PRINCIPAL**

**print("Metodos return")**

**print("Introduzca un texto")**

**valor = input()**

**mostrarMenu()**

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

**resultado = ""**

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

**resultado = convertirMayusculas(valor)**

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

**resultado = convertirMinusculas(valor)**

**else:**

**print("Ponga otro texto")**

**otro = input()**

**resultado = concatenar(valor, otro)**

**print(resultado)**

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

Creamos un nuevo programa **python23calculadora.py**

1. Pediremos al usuario dos números
2. Debemos mostrar un menú con estas operaciones:

1.- Sumar

2.- Restar

3.- Multiplicar

Tendremos tres funciones/métodos **return** para cada una de las opciones y mostraremos el resultado de lo que el usuario ha seleccionado

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

VERSION 1

**# DECLARACION METODOS**

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

**return num1 + num2**

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

**return num1 - num2**

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

**return num1 \* num2**

***def* mostrarMenu():**

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

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

**print("3.- Multiplicar")**

**print("Seleccione una opción")**

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

**print("Calculadora metodos")**

**print("Introduzca numero 1")**

**numero1 = *int*(input())**

**print("Introduzca numero 2")**

**numero2 = *int*(input())**

**mostrarMenu()**

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

**operacion = 0**

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

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

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

**operacion = restarNumeros(numero1, numero2)**

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

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

**else:**

**print("No ha seleccionado una opción correcta")**

**print("Operación " + *str*(operacion))**

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

Versión 2: Realizamos el mismo ejemplo, pero vamos a utilizar un bucle para que aparezca el menú en todo momento hasta que el usuario seleccione la opción que yo quiera.

**VERSION 2**

**# DECLARACION METODOS**

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

**return num1 + num2**

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

**return num1 - num2**

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

**return num1 \* num2**

***def* mostrarMenu():**

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

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

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

**print("3.- Multiplicar")**

**print("Seleccione una opción")**

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

**print("Calculadora metodos")**

**print("Introduzca numero 1")**

**numero1 = *int*(input())**

**print("Introduzca numero 2")**

**numero2 = *int*(input())**

**# ASIGNAMOS UN VALOR A OPCION PARA ENTRAR EN EL BUCLE**

**opcion = 1**

**# CREAMOS UN WHILE HASTA QUE EL USUARIO ESCRIBA 0**

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

**mostrarMenu()**

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

**operacion = 0**

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

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

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

**operacion = restarNumeros(numero1, numero2)**

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

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

**else:**

**print("No ha seleccionado una opción correcta")**

**print("Operación " + *str*(operacion))**

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

VERSION 3.

* Si el usuario no introduce números (inicio) volver a pedirle números hasta que nos de dos números
* Necesitamos una opción más para que el usuario pueda introducir números de nuevo si lo desea

**# DECLARACION METODOS**

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

**return num1 + num2**

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

**return num1 - num2**

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

**return num1 \* num2**

***def* mostrarMenu():**

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

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

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

**print("3.- Multiplicar")**

**print("4.- Introducir numeros de nuevo")**

**print("Seleccione una opción")**

***def* getNumeroComprobado():**

**print("Introduzca numero")**

**# ALMACENAR LO QUE HA ESCRITO EL USUARIO**

**# EN UNA VARIABLE STRING**

**aux = input()**

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

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

**print("Introduzca numero")**

**aux = input()**

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

**return num**

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

**print("Calculadora metodos")**

**numero1 = getNumeroComprobado()**

**numero2 = getNumeroComprobado()**

**# ASIGNAMOS UN VALOR A OPCION PARA ENTRAR EN EL BUCLE**

**opcion = 1**

**# CREAMOS UN WHILE HASTA QUE EL USUARIO ESCRIBA 0**

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

**mostrarMenu()**

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

**operacion = 0**

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

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

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

**operacion = restarNumeros(numero1, numero2)**

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

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

**elif (opcion == 4):**

**# DEBEMOS PEDIR NUMEROS**

**numero1 = getNumeroComprobado()**

**numero2 = getNumeroComprobado()**

**elif (opcion == 0):**

**print("Hasta luego")**

**else:**

**print("No ha seleccionado una opción correcta")**

**print("Operación " + *str*(operacion))**

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

# LIBRERIAS EN PYTHON

Python nos permite tener los métodos que deseemos en librerías, que no dejan de ser ficheros Python, pero que no se ejecutan, sino que son llamados por otros ficheros Python.

Por ejemplo, hemos utilizado la clase **math**, que es una librería que contiene una serie de métodos. Dicha librería en ningún momento la hemos ejecutado, sino que la hemos utilizado.

Ejemplo: **math.py**

def trunc(6.7):

….

def floor(5.9):

….

Lo que seguiremos haciendo a partir de ahora es tener un fichero PY que será el principal (main/program): **program.py**

Tendremos librerías PY con solo métodos para ser llamados. **libreria.py**

**LIBRERÍA.PY**

def metodo1():

def metodo2():

**PROGRAM.PY**

from **librería.py** import metodo1(), metodo2()

metodo1()

import librería.py

librería.metodo1()

**Nota: Los nombres de las librerías de Python serán el nombre de los ficheros Python**

Vamos a crear un nuevo fichero Python llamado **libreria.py**

Ahora vamos a trabajar con 2 clases:

* libreriaXX: La clase con los métodos
* pythonXX: La clase que ejecutamos

Vamos a crear el ejemplo de la calculadora, pero llevando solo lo necesario (sumar, restar, menú) y lo llamamos desde otra clase.

Creamos un nuevo fichero llamado **libreria24matematicas.py**

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

**return num1 + num2**

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

**return num1 - num2**

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

**return num1 \* num2**

***def* mostrarMenu():**

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

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

**print("3.- Multiplicar")**

**print("Seleccione una opción")**

Creamos un nuevo programa llamado **python24matematicas.py**

**# LOS IMPORT SE REALIZAN LO PRIMERO DE NUESTRO CODIGO**

**#from libreria24matematicas import sumarNumeros, restarNumeros, mostrarMenu**

**import libreria24matematicas**

**# CODIGO LOGIGO**

**print("Calculadora métodos")**

**numero1 = 9**

**numero2 = 19**

**libreria24matematicas.mostrarMenu()**

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

**resultado = 0**

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

**resultado = libreria24matematicas.sumarNumeros(numero1, numero2)**

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

**resultado = libreria24matematicas.restarNumeros(numero1, numero2)**

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

**resultado = libreria24matematicas.multiplicarNumeros(numero1, numero2)**

**else:**

**print("No ha seleccionado una opción correcta")**

**print("Resultado ", resultado)**

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

En el siguiente ejemplo lo que tenemos que hacer es ir copiando código y adaptarlo a métodos y llamadas.

Un método es un resultado si deseamos recuperar algo que no sea un dibujo de un menú.

Vamos a crear lo siguiente:

* **libreria25validaciones.py**

1. Tendremos un método que validará un ISBN, recibirá un ISBN y nos devolverá si es correcto o no. (True/False)
2. Necesitamos un método que nos devolverá la letra del DNI. Enviamos un número de DNI y nos devuelve la letra que corresponda.

***def* validarDni(*numeroDni*):**

**resultado = numeroDni % 23**

**letra = ""**

**if (resultado == 0):**

**letra = "T"**

**elif (resultado == 1):**

**letra = "R"**

**elif (resultado == 2):**

**letra = "W"**

**elif (resultado == 3):**

**letra = "A"**

**elif (resultado == 4):**

**letra = "G"**

**elif (resultado == 5):**

**letra = "M"**

**elif (resultado == 6):**

**letra = "Y"**

**elif (resultado == 7):**

**letra = "F"**

**elif (resultado == 8):**

**letra = "P"**

**elif (resultado == 9):**

**letra = "D"**

**elif (resultado == 10):**

**letra = "X"**

**elif (resultado == 11):**

**letra = "B"**

**elif (resultado == 12):**

**letra = "N"**

**elif (resultado == 13):**

**letra = "J"**

**elif (resultado == 14):**

**letra = "Z"**

**elif (resultado == 15):**

**letra = "S"**

**elif (resultado == 16):**

**letra = "Q"**

**elif (resultado == 17):**

**letra = "V"**

**elif (resultado == 18):**

**letra = "H"**

**elif (resultado == 19):**

**letra = "L"**

**elif (resultado == 20):**

**letra = "C"**

**elif (resultado == 21):**

**letra = "K"**

**elif (resultado == 22):**

**letra = "E"**

**elif (resultado == 23):**

**letra = "T"**

**return letra**

***def* validarISBN(*isbn*):**

**longitud = len(isbn)**

**if (longitud != 10):**

**return False**

**elif (isbn.isdigit() == False):**

**return False**

**else:**

**suma = 0**

**for i in range(longitud):**

**letra = isbn[i]**

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

**operacion = numero \* (i + 1)**

**suma = suma + operacion**

**if (suma % 11 == 0):**

**return True**

**else:**

**return False**

* **python25validaciones.py**

1. Pedimos al usuario un ISBN y comprobamos el método
2. Pedimos al usuario su número de DNI y comprobamos el método

**PYTHON25VALIDACIONES.PY**

**from libreria25validaciones import validarISBN, validarDni**

**print("Clase Program validaciones")**

**print("Introduzca un número DNI")**

**numeroDni = *int*(input())**

**letra = validarDni(numeroDni)**

**print("Letra " + letra)**

**print("----------------")**

**print("Introduzca ISBN")**

**isbn=input()**

**valido = validarISBN(isbn)**

**print("El isbn es " , valido)**

## COMO INSTALAR PAQUETES

Un paquete es un conjunto de librerías que podemos utilizar en múltiples equipos.

Ahora mismo nuestras librerías están en este equipo, si quisiéramos utilizarlas en cualquier otro proyecto de Python de lo que sea, no podemos hacerlo.

Existe un repositorio de librerías para Python.

Dicho repositorio contiene todas las librerías de terceros hacia Python con sus actualizaciones y todo.

Esto nos puede servir, por ejemplo, si deseamos leer una base de datos SQL Server. Necesitamos una librería, de dónde la sacamos?

Si necesitamos leer una base de datos Oracle, de donde sacamos la librería?

Para instalar librerías directamente necesitamos Python superior a 3.13

Cuando instalamos Python, se instala un paquete llamado **pip** que es el encargado de descargar las librerías para que yo pueda utilizarlas dentro de mi equipo.

El comando para descargar es (Windows): **pip install NOMBRELIBRERIA**

Vamos a instalar una librería para generar ejecutables del S.O

Esta librería genera un EXE si estamos en Windows o un PKG si estamos en macOS.

Abrimos Terminal en Mac

La librería que vamos a instalar se llama **pyinstaller**

**pip3 install pyinstaller**

Mediante pyinstaller ya podemos crear ejecutables en nuestro equipo a partir de programas PY

Buscamos un programa PY autónomo, es decir, que no tire de una librería de las nuestras

Por ejemplo, el de Collatz y lo copiamos a la carpeta de Documents.

Vamos a crear el ejecutable, mediante la línea de Terminal nos posicionamos dónde tengamos el PY.

ls Vemos las carpetas

cd Carpeta. Entramos en una carpeta

**pyinstaller -c -F NOMBREPYTHON.py**

Cuando hablamos de objetos conjuntos, es decir, hablamos de LISTAS, TUPLAS o DICCIONARIOS tenemos un bucle para realizar un recorrido por cada objeto.

A este tipo de bucles se le llama **Bucle de referencia** y, en la declaración del bucle, se utiliza una variable como referencia de cada elemento que estamos recorriendo.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Nos permite declarar **prod** dentro del Bucle en lugar de **i**

for VARIABLE in CONJUNTO:

variable

**EJEMPLO LISTAS**

**print("Listas con Python")**

**listaNumeros = [12,56,77,88, 1,99,22]**

**# ORDEN ASCENDENTE**

**# listaNumeros.sort()**

**# Para ordenar de forma descendente debemos incluir**

**# dentro del método reverse = True**

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

**# REALIZAMOS UN BUCLE PARA MOSTRAR LOS NUMEROS ACTUALMENTE**

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

**print(listaNumeros[i])**

**# LAS LISTAS COMIENZAN EN CERO Y FINALIZAN EN LEN -1**

**print("Numero 0: ", listaNumeros[0])**

**print("Numero 1: ", listaNumeros[1])**

**listaNombres = ["Ana", "Lucas", "Adrian", "Diana", "Antonia", "Lucas"]**

**print("Nombre 2: ", listaNombres[2])**

**print("Nombre 4: ", listaNombres[4])**

**# append CREA UN NUEVO ELEMENTO EN LA LISTA AL FINAL**

**listaNombres.append("Lucia")**

**print("Nombre 5: ", listaNombres[5])**

**# insert() CREA UN ELEMENTO NUEVO EN UNA POSICION DE LA LISTA**

**listaNombres.insert(4, "Infiltrado")**

**# El metodo remove() elimina el primer objeto dentro de la lista**

**# si no lo encuentra da error**

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

**#listaNombres.pop(6)**

**# del listaNombres[0:4]**

**# clear() borra todo el contenido de una lista**

**# listaNombres.clear()**

**# Vamos a recorrer todos los elementos de la lista y mostrar**

**# su posición**

**# print("Dianis" in listaNombres)**

**# print("elementos")**

**listaNombres.sort()**

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

**print(*str*(i) + "=" + listaNombres[i])**

**EJEMPLO TUPLAS**

**print("Tuplas de Python")**

**# Las Tuplas no se pueden modificar**

**productos = ("Leche", "Cacao", "Avellanas", "Azucar")**

**#productos[2] = "Havellanbas"**

**numeroElementos = len(productos)**

**# print("Elementos tupla ", numeroElementos)**

**for producto in productos:**

**print(producto)**

Necesito poder almacenar la información de un Mes

Un mes tiene un nombre, una temperatura máxima y una temperatura mínima

En el mes podemos guardar tanto la máxima como la mínima y su nombre

Necesito calcular la media del mes (que se calcule sola y me la devuelva)

Al escribir el objeto mes (**print(mes)**) me gustaría visualizar el nombre, la máxima y la mínima directamente.

Necesitamos dos clases: class32mes.py y el python32mes.py

En el Python, creamos dos meses y lo probamos.

**MES**

***class* Mes:**

**nombre = ""**

**temperaturaMaxima = 0**

**temperaturaMinima = 0**

***def* getTemperaturaMedia(*self*):**

**return (*self*.*temperaturaMaxima* + *self*.*temperaturaMinima*) / 2**

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

**return *self*.*nombre* + ", Max: " + *str*(*self*.*temperaturaMaxima*) + ", Min: " + *str*(*self*.*temperaturaMinima*)**

PYTHON32MES

**from class32mes import Mes**

**print("Trabajando con clase Mes")**

**enero = Mes()**

**enero.*nombre* = "Enero"**

**enero.*temperaturaMaxima* = 9**

**enero.*temperaturaMinima* = -4**

**media = enero.getTemperaturaMedia()**

**print("Enero ", media)**

**print(enero)**

**febrero = Mes()**

**febrero.*nombre* = "Febrero"**

**febrero.*temperaturaMaxima* = 18**

**febrero.*temperaturaMinima* = 4**

**media = febrero.getTemperaturaMedia()**

**print("febrero ", media)**

**print(febrero)**

El siguiente paso será tener muchos meses, pero intentaremos hacerlo de una forma sencilla.

Como no quiero estar pidiendo nada al usuario, vamos a automatizar ciertas cosas como, por ejemplo, la Máxima y la mínima.

Tenemos una clase llamada **random** que sirve para generar números aleatorios.

La clase random tiene un método para recuperar números decimales y otro para recuperar números enteros.

**from class32mes import Mes**

**import random**

**print("Trabajando con clase Mes")**

**meses = ("Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre")**

**for nombreMes in meses:**

**mes = Mes()**

**mes.*nombre* = nombreMes**

**mes.*temperaturaMaxima* = random.randint(1, 40)**

**mes.*temperaturaMinima* = random.randint(1, 40)**

**media = mes.getTemperaturaMedia()**

**print(media)**

**print(mes)**

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

INSTALACION ORACLE XE

Oracle es una base de datos y tenemos una pequeña edición para poder trabajar con Oracle.

En todas las bases de datos empresariales suelen tener una edición para jugar, es decir, sin coste.

Dichas bases de datos se les denomina Express

En Oracle la base de datos gratuita se llama Oracle XE

En Windows, lo único que necesitamos es espacio en disco duro y tiempo. La instalación tarda bastante y creo que ocupa unos 4 gb

Lo bueno es que es siguiente y siguiente

<https://www.oracle.com/es/database/technologies/xe-downloads.html>

En Mac no tenemos una edición para trabajar con estos equipos, debemos hacer una instalación mediante Docker.

Un Docker es una herramienta que permite crear funcionalidades, como máquinas virtuales, sin necesidad de instalaciones.

Para instalar en estos Mac, necesitamos esta guía.

<https://fjtoscano.medium.com/instalar-oracle-database-xe-en-mac-m1-d5d2d17fc00c>

Entramos en HomeBrew

<https://brew.sh>

Interfaz de usuario gráfica, Texto, Sitio web

El contenido generado por IA puede ser incorrecto.

Abrimos **Terminal** y copiamos ahí este código

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

Cuando lo tengamos, nos aparecerá al final un mensaje con la instrucción **echo** para escribir en el **Terminal**

**brew install colima**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Ejecutamos la instrucción que nos indica

**brew services start colima**

Instalamos Docker

**brew install docker**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Instalamos **qemu**

**brew install qemu**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Ejecutamos la instrucción de **colima start**

colima start --arch x86\_64 --memory 4

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Ya tenemos todo montado.

A continuación, mediante terminal, descargamos la imagen Docker de Oracle para trabajar

docker pull gvenzl/oracle-xe

Una vez descargada nuestra imagen, para poder iniciar un Container con esta imagen y que nos podamos comunicar con ella, debemos ejecutar esta instrucción.

En dicha instrucción, tendremos que poner una contraseña para acceder a Oracle.

Vamos a poner como contraseña: **oracle**

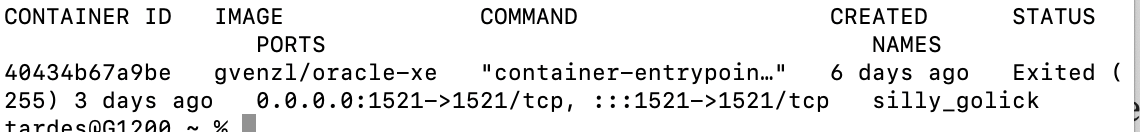
docker run -d -p 1521:1521 -e ORACLE\_PASSWORD=**oracle** -v oracle-volume:/opt/oracle/oradata gvenzl/oracle-xe

Ya tenemos montado Oracle en nuestra máquina.

El Container tendrá un ID único que nos servirá para arrancarlo cada día sin tener que poner nada más.

Para visualizar nuestros Containers escribimos la siguiente instrucción en el Terminal

docker container ls -a



Necesitamos nuestro **CONTAINER ID** para arrancarlo cada día.

Cada día, arrancamos nuestro Oracle en Docker mediante la siguiente instrucción

docker container start **CONTAINER ID**

Texto, Carta

El contenido generado por IA puede ser incorrecto.

Una vez que temenos Oracle XE necesitamos de un IDE para poder realizar y ejecutar consultas sobre la base de datos.

Vamos a utilizar el oficial: **SQL Developer Oracle**

<https://www.oracle.com/es/database/sqldeveloper/technologies/download/>

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Para casa, tenemos una herramienta que no consume tantos recursos como el programa **Developer de Oracle**

Dicha herramienta sirve para lo mismo y consume mucho menos espacio y recursos

<https://dbeaver.io/download/>

A continuación, vamos a realizar una conexión a nuestro Oracle XE

Para ello, simplemente lo haremos una vez y siempre quedará guardada en nuestro entorno de trabajo

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

El contenido generado por IA puede ser incorrecto.

Esta base de datos tiene un único inconveniente en el que solamente podemos tener una única base de datos. XE

Usuario Oracle: **SYSTEM**

Password: **oracle**

Server: **LOCALHOST**

Base de datos: **XE**

Puerto: **1521**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Una vez conectados, copiamos el script y lo pegamos dentro de Oracle Developer

Para ejecutar TODO el script utilizamos la tecla F5 o la hoja de papel con el icono verde

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

El contenido generado por IA puede ser incorrecto.

Una vez que tenemos todo montado, podemos comenzar.

LENGUAJE ANSI SQL

* Es un estándar de 1992 en el que indica que todas las bases de datos utilizarán la misma sintaxis para recuperar y trabajar con los datos.
* Esto no quiere decir que cada base de datos, además, tenga otro tipo de consultas distintas para sus funciones.
* Dentro de estas consultas podemos dividir las consultas entre las siguientes:
  + Consultas de selección
    - Consultas simples
    - Consultas de agrupación
    - Consultas de combinación
    - Subconsultas
    - Nivel de fila y Select to Select
  + Consultas de acción
    - Insert
    - Update
    - Delete

LENGUAJE PL/SQL

Cada base de datos Empresarial tiene un lenguaje propio para trabajar. Estoy hablando de IF y BUCLES pero adaptado a los datos.

Dicho lenguaje, en Oracle se llama PL/SQL

Este lenguaje está orientado a objetos propios de la base de datos:

* Views: Consultas almacenadas dentro de la base de datos
* Stored Procedures: Procedimientos de código almacenados en base de datos
* Package: Estructuras para organizar objetos

El editor de PL/SQL nos permite tener todas las consultas que deseemos escritas

Solamente las consultas que seleccionemos se ejecutarán

Texto

El contenido generado por IA puede ser incorrecto.

La consulta más standard que existe es la básica:

Devuelve todos los registros y todas las columnas de una tabla

select \* from TABLA

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

También podemos indicar las columnas que deseemos separadas por coma dentro del select

select column1, columna2, columna3 from TABLA

select DEPT\_NO, DNOMBRE, LOC from DEPT;

Dentro de bases de datos, tendremos consultas que nos devolverán la misma información pero habrá algunas consultas más eficaces (rápidas en ejecución) que otras.

**CONSULTAS BASICAS ANSI SQL**

-- COMENTARIOS

/\*

varias lineas

PODEMOS TENER TODAS LAS CONSULTAS QUE DESEEMOS

NO ES NECESARIO BORRARLAS

EN EL EDITOR, INDICAREMOS QUE CONSULTA (LINEA O SELECCION)

QUEREMOS EJECUTAR

PARA EJECUTAR CONTROL + ENTER (MAC)

NO DIFERENCIA MAYUSCULAS DE MINUSCULAS EN LAS CONSULTAS

LAS CONSULTAS FINALIZAN CON ;

\*/

select \* from DEPT;

select DEPT\_NO, DNOMBRE, LOC from DEPT;

--ORDENAR REGISTROS order by COLUMNA

select \* from EMP order by APELLIDO desc;

--FILTRADO DE REGISTROS

--PARA FILTRAR UTILIZAMOS UNA PALABRA LLAMADA where

/\*

OPERADORES DE COMPARACION

> MAYOR

>= MAYOR O IGUAL

< MENOR

<= MENOR O IGUAL

= IGUAL

<> DISTINTO

TODO LO QUE NO SEA UN NUMERO SE ESCRIBE ENTRE COMILLAS SIMPLES

PARA LA COMPARACION

ORACLE DIFERENCIA MAYUSCULAS DE MINUSCULAS EN LOS STRING

\*/

select \* from EMP where DEPT\_NO=10;

--Mostrar todos los empleados cuyo oficio sea DIRECTOR

select \* from EMP where OFICIO='DIRECTOR';

select \* from EMP where OFICIO='director';

/\*

OPERADORES RELACIONALES

PERMITEN PREGUNTAR POR MAS DE UN FILTRO EN UNA MISMA CONSULTA

and TODAS LAS CONDICIONES DEBEN CUMPLIRSE

or MUESTRA DATOS DE CADA CONDICION

not PROHIBIDO, NO LO DEBEMOS UTILIZAR. NIEGA UNA CONDICION

\*/

--Mostrar todos los empleados del departamento 10 y del departamento 20

select \* from EMP where DEPT\_NO=10 or DEPT\_NO=20;

--Mostrar todos los empleados cuyo EMP\_NO este entre 7800 y 7900

select \* from EMP where EMP\_NO >= 7800 and EMP\_NO <= 7900;

--Mostrar todos los empleados que NO sean vendedores

select \* from EMP where not OFICIO='VENDEDOR';

select \* from EMP where OFICIO <> 'VENDEDOR';

--EXISTEN OTROS TIPOS DE OPERADORES PARA FILTRAR

--OPERADOR BETWEEN: BUSCA ENTRE UN RANGO Y DEVUELVE LOS VALORES

--INCLUSIVE DEL RANGO

--Mostrar los empleados cuyo salario esté entre 318000 y 390000

select \* from EMP where SALARIO between 318000 and 390000;

select \* from EMP where SALARIO >= 318000 and SALARIO <= 390000;

--OPERADOR IN

--BUSCA EN UN MISMO CAMPO MULTIPLES VALORES DE IGUALDAD

--Mostrar todos los empleados del departamento 10 y 20, 30, 55,66,88

-- CAMPO IN (VALOR1, VALOR2, VALOR3)

select \* from EMP where DEPT\_NO=10 or DEPT\_NO=20 or DEPT\_NO=30

or DEPT\_NO=55 or DEPT\_NO=66 or DEPT\_NO=88;

select \* from EMP where DEPT\_NO in (10,20,30,55,66,88);

--OPERADOR NOT IN

--Muestra los contrarios al operador IN

--Mostrar todos los empleados que no estén en el departamento 10 ni en el 20

select \* from EMP where DEPT\_NO NOT in (10, 20);

--OPERADOR LIKE

--SE UTILIZA PARA BUSCAR COINCIDENCIAS EN TEXTOS (STRING, VARCHAR)

--BUSCADORES

--UTILIZA UNA SERIE DE CARACTERES ESPECIALES (SOLO SIRVE UNO)

-- ? Busca un digito en el texto

-- \_ Representa un caracter cualquiera

-- % Busca cualquier caracter y cualquier longitud

--Mostrar todos los empleados cuyo apellido comience con S

select \* from EMP where apellido like 's%';

--Mostrar todos los empleados cuyo apellido comience con S y finalicen con A

select \* from EMP where apellido like 's%a';

--Mostrar todos los empleados cuyo apellido sea de 4 letras

select \* from EMP where apellido like '\_\_\_\_';

--CLAUSULA DISTINCT

--Nos permite quitar repetidos de una consulta SELECT

--En realidad, se utiliza con pocos campos.

--Mostrar los oficios de los empleados

select distinct OFICIO from EMP;

--CAMPOS CALCULADOS

--Un campo calculado es una columna que NO existe en una tabla

--pero se genera a partir de otra columna/s

--OPERADORES MATEMATICOS: +, \*, /, -

--TODO CAMPO CALCULADO DEBE TENER UN ALIAS SIEMPRE

--LOS ALIAS SIN ESPACIOS

--Mostrar el apellido y el salario Total (salario + comision) de todos los empleados.

select APELLIDO, (SALARIO + COMISION) as TOTAL from EMP;

--LOS CAMPOS CALCULADOS NO SE PUEDEN FILTRAR CON where

--UN where SOLAMENTE SE UTILIZA PARA FILTRAR DATOS DE LA TABLA

--SI DESEAMOS FILTRAR SOBRE UN CAMPO CALCULADO, DEBEMOS HACERLO

--CON EL MISMO CALCULO

select APELLIDO, (SALARIO + COMISION) as TOTAL from EMP

where (SALARIO + COMISION) >= 210000;

--UN order by SE EJECUTA SOBRE EL CURSOR

select APELLIDO, (SALARIO + COMISION) as TOTAL from EMP order by TOTAL asc;

# CONSULTAS DE AGRUPACION

Son consultas que devuelven información sobre un determinado grupo.

Nunca nos devolverán los elementos que corresponden a dicho grupo, en realidad, se utiliza para poder mostrar algún resumen de los datos que tenemos.

Se utilizan funciones de agrupación:

* COUNT(\*): Cuenta el número de registros de un grupo. Tiene en cuenta los nulos
* COUNT(CAMPO): Cuenta el número de registros de un grupo. No tiene en cuenta los nulos
* MAX(CAMPO): Devuelve el máximo valor de un campo
* MIN(CAMPO): Devuelve el mínimo valor de un campo
* AVG(NUMERO): Devuelve la media de un conjunto
* SUM(NUMERO): Devuelve la suma de un conjunto

Estas funciones se pueden combinar entre sí en la misma consulta

Las columnas con funciones deben llevar siempre ALIAS

--Contar el número de registros de la tabla EMP

select COUNT(\*) as REGISTROS from EMP;

--Mostrar el número de registros y el máximo salario de la tabla EMP

select COUNT(\*) as REGISTROS, MAX(SALARIO) as MAXIMO from EMP;

Se llaman consultas de agrupación porque podemos agrupar el resumen por columnas.

Por ejemplo, imaginemos que deseamos mostrar el número de personas por cada oficio.

La agrupación de columnas se realiza en el SELECT.

Debemos indicar, mediante **GROUP BY**, cada columna por la que estamos agrupando.

**Truco:** Debemos incluir en el group by cada columna que no sea un función de agrupación en el select.

**EJEMPLOS**

--Mostrar el número de personas por cada oficio

select COUNT(\*) as PERSONAS, OFICIO from EMP

group by OFICIO;

--Mostrar el número de personas por cada oficio y departamento

select COUNT(\*) as PERSONAS, OFICIO, DEPT\_NO

, SUM(SALARIO) as SUMA from EMP

group by OFICIO, DEPT\_NO;

Podemos perfectamente filtrar estas consultas.

Para filtrar tenemos dos opciones:

1. **Where:** Filtra por los datos de la tabla. Antes del GROUP
2. **Having:** Filtra por los datos del grupo. Después del GROUP

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Podemos hacer lo mismo con HAVING (más rápido)

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**Norma**: Si estoy filtrando por algún campo de la consulta SELECT con GROUP BY, es más rápido utilizar HAVING

En otras ocasiones, no tengo elección. Si quiero filtrar por una columna de la tabla que no esté en el SELECT, tengo que hacerlo con **where**

Texto

El contenido generado por IA puede ser incorrecto.

Mostrar el número de personas por cada oficio pero solamente los oficios dónde tengamos más de una persona trabajando.

Texto

El contenido generado por IA puede ser incorrecto.

# CONSULTAS DE COMBINACION

Muy importantes. Permiten representar datos de múltiples tablas relacionadas entre sí en un mismo cursor de resultados.

Las tablas deben tener un campo de relación entre sí para este tipo de consultas.

Se realizan estas consultas cuando queremos combinar múltiples campos para mostrarlos directamente en pantalla.

SINTAXIS:

select TABLA1.CAMPO1, TABLA1.CAMPO2

, TABLA2.CAMPO1, TABLA2.CAMPO2

from TABLA1

inner join TABLA2

on TABLA1.CAMPORELACION = TABLA2.CAMPORELACION

**Nota: No importa el orden de TABLA1 o TABLA2 en ningún sitio.**

Ejemplos:

--Mostrar el Apellido, Oficio de los empleados junto a

--su nombre de departamento y localidad.

select EMP.APELLIDO, EMP.OFICIO

, DEPT.DNOMBRE, DEPT.LOC

from EMP

inner join DEPT

on EMP.DEPT\_NO=DEPT.DEPT\_NO;

--Mostrar el Apellido, Oficio, nombre de departamento y localidad

--de los empleados de SEVILLA

select EMP.APELLIDO, EMP.OFICIO

, DEPT.DNOMBRE, DEPT.LOC

from EMP

inner join DEPT

on EMP.DEPT\_NO=DEPT.DEPT\_NO

where DEPT.LOC='SEVILLA';

Ejemplo con consultas de AGRUPACION

--Por supuesto, podemos combinar esto junto a funciones de agrupación

--Queremos mostrar el número de personas por cada número de departamento.

select count(\*) as PERSONAS, DEPT\_NO

from EMP

group by DEPT\_NO;

--Queremos mostrar el número de personas por cada NOMBRE de departamento

select COUNT(\*) as PERSONAS, DEPT.DNOMBRE

from EMP

inner join DEPT

on EMP.DEPT\_NO=DEPT.DEPT\_NO

group by DEPT.DNOMBRE;

**Ejemplo**

--Mostrar el Apellido, función y NOMBRE DE HOSPITAL

--de las personas de la plantilla

--1) ABRIR LA ESTRUCTURA DE LAS TABLAS Y MIRAR CON QUE SE RELACIONA PLANTILLA

--2) HACER LA CONSULTA

select PLANTILLA.APELLIDO, PLANTILLA.FUNCION

, HOSPITAL.NOMBRE

from PLANTILLA

inner join HOSPITAL

on PLANTILLA.HOSPITAL\_COD=HOSPITAL.HOSPITAL\_COD;

Tenemos, dentro de las consultas de Combinación otro tipo de consultas dependiendo lo que necesitemos recuperar.

* INNER JOIN: Recupera los datos que combinen entre las dos tablas.
* LEFT JOIN: Recupera los datos que combinen y también los datos que no combinan de la tabla de la izquierda.
* RIGHT JOIN: Recupera los datos que combinen y también los datos que no combinan de la tabla de la derecha.
* FULL JOIN: Recupera todos los datos, combinen o no.
* CROSS JOIN: Recupera el producto cartesiano, la combinación de cada fila de una tabla con las diferentes posibilidades de cada fila de la otra tabla.

En todas las bases de datos relacionales, existe algo llamado **integridad relacional** que hace que los datos estén correctos y que todo esté en orden.

Normalmente, en dicha integridad, lo que hacemos es crear unas NORMAS para que los datos estén relacionados entre sí.

Actualmente, tenemos un **Departamento** sin **EMPLEADOS (40, PRODUCCION, GRANADA)**

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Para comprobar la funcionalidad, vamos a insertar un Empleado que no tenga Departamento

1111, GETAFE, ESTUDIANTE, 1, FECHA, 1, 1, **50**

Texto, Tabla

El contenido generado por IA puede ser incorrecto.

Veremos que INNER JOIN no muestra ni GETAFE ni GRANADA

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.

La sintaxis de LEFT JOIN o de RIGHT JOIN no cambia.

Solamente tenemos que saber lo siguiente:

* TABLA IZQUIERDA (LEFT): Antes del JOIN
* TABLA DERECHA (RIGHT): Después del JOIN

Si utilizamos ahora un LEFT JOIN estamos mostrando los Empleados aunque no tengan departamento. (GETAFE)

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Texto, Tabla

El contenido generado por IA puede ser incorrecto.

Si utilizamos RIGHT JOIN veremos los **Departamentos** aunque no tengan empleados (GRANADA)

Texto

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

El siguiente es FULL JOIN que muestra todos los datos, combinen o no combinen.

Texto

El contenido generado por IA puede ser incorrecto.

Veremos datos de las dos tablas (GRANADA y GETAFE)

Texto

El contenido generado por IA puede ser incorrecto.

CROSS JOIN muestra el producto Cartesiano. No lleva **ON**

Texto

El contenido generado por IA puede ser incorrecto.

* Quiero visualizar el Apellido, Salario y Especialidad junto a la dirección del hospital donde trabaja de todos los doctores de La paz.

select DOCTOR.APELLIDO, DOCTOR.SALARIO, DOCTOR.ESPECIALIDAD

, HOSPITAL.DIRECCION

from DOCTOR

inner join HOSPITAL

on DOCTOR.HOSPITAL\_COD=HOSPITAL.HOSPITAL\_COD

where HOSPITAL.NOMBRE='la paz';

* Quiero visualizar cuantos empleados trabajan en cada Departamento mostrando el nombre del Departamento.

Esta consulta solamente estará mostrando los datos de los Departamentos que tengan empleados y nosotros queremos todos.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

La consulta correcta:

select COUNT(EMP.EMP\_NO) as EMPLEADOS

, DEPT.DNOMBRE

from EMP

right join DEPT

on EMP.DEPT\_NO=DEPT.DEPT\_NO

group by DEPT.DNOMBRE;

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Por supuesto, las consultas se pueden complicar.

Un Join siempre estará compuesto por dos o más tablas y puede ser que tenga que unir 10 tablas relacionadas.

Si tenemos más de una tabla, simplemente debemos incluir INNER JOIN y ON por cada tabla extra que tengamos en la consulta.

Sintaxis:

select TABLA1.CAMPO1, TABLA1.CAMPO2

, TABLA2.CAMPO1, TABLA2.CAMPO2

, TABLA3.CAMPO1, TABLA3.CAMPO2

from TABLA1

inner join TABLA2

on TABLA1.CAMPORELACION=TABLA2.CAMPORELACION

inner join TABLA3

on TABLA1.CAMPORELACION=TABLA3.CAMPORELACION;

Con esta consulta ya lo tendríamos normalmente en cualquier base de datos.

Pero, en esta base de datos, la PLANTILLA se relaciona a su vez con la SALA.

Por lo que también tendríamos que quitar la relación entre SALA y PLANTILLA con su campo SALA\_COD.

En cualquier consulta SELECT no podemos tener nombres de columnas repetidos, deberíamos poner un ALIAS si deseamos trabajar con los datos.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Si tenemos más de una relación que quitar con el ON, se utiliza el operador **AND**

select PLANTILLA.APELLIDO, PLANTILLA.FUNCION

, HOSPITAL.NOMBRE as HOSPITAL

, SALA.NOMBRE as SALA

from PLANTILLA

inner join HOSPITAL

on PLANTILLA.HOSPITAL\_COD=HOSPITAL.HOSPITAL\_COD

inner join SALA

on SALA.HOSPITAL\_COD=HOSPITAL.HOSPITAL\_COD

and SALA.SALA\_COD=PLANTILLA.SALA\_COD;

# SUBCONSULTAS

Son muy útiles en determinados momentos.

En realidad, son consultas en las que necesitamos el dato de otra consulta para poder ser ejecutada.

En las consultas de selección deberíamos intentar no utilizarlas, ya que generan bloqueos, pero si no se puede, no se puede.

No importa el nivel de anidamiento en las consultas.

Necesito mostrar los datos del empleado que más cobra de la Empresa.

¿Qué pasos necesitamos?

Valor máximo de la columna salario: 1 Consulta

Hacer otra consulta con ese valor que nos ha devuelto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

En lugar de estar copiando resultados e igualando dichos resultados (650000) a cada consulta posterior, las subconsultas se ejecutan **anidando** unas con otras y ejecutando todas a las vez.

Sintaxis:

Select \* from PRIMERACONSULTA where dato=(select … from SEGUNDACONSULTA);

Ejemplo

Texto

El contenido generado por IA puede ser incorrecto.

--Mostrar todos los empleados que tengan el mismo oficio que SALA

--y que tengan más salario que el empleado FORD

--Oficio de Sala: VENDEDOR

--Salario FORD: 162500

select \* from EMP where OFICIO='VENDEDOR' and SALARIO > 162500;

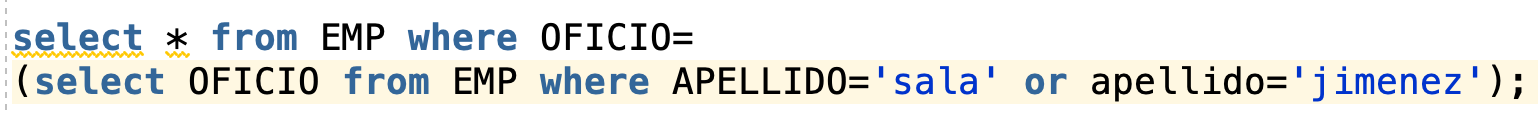
select \* from EMP where OFICIO=(select OFICIO from EMP where APELLIDO='sala')

and SALARIO > (select SALARIO from EMP where APELLIDO='ford');

Cuando una subconsulta devuelve más de un valor, nos dará una excepción porque NO SABE a que valor establecer la igualdad.

**Mostrar todos los empleados que tengan el mismo oficio que SALA o que tengan el mismo oficio que JIMENEZ**

Ejecutando esta consulta, podremos visualizar que nos está dando un error

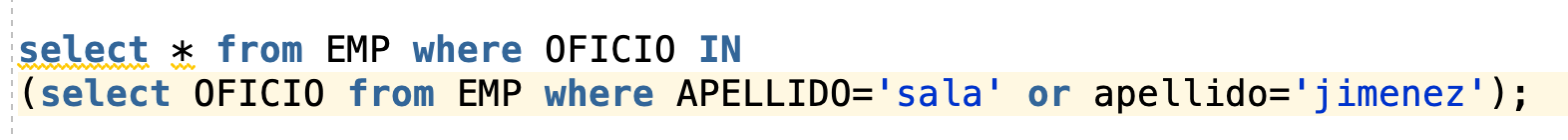


Podremos comprobar que la subconsulta nos está devolviendo dos registros:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Cuando una subconsulta devuelve más de un valor se utiliza el operador **IN**



# CONSULTAS UNION

Estas consultas sirven para mostrar datos de varias tablas que **NO** tienen relación dentro de un mismo CURSOR.

Tenemos que seguir tres reglas:

1. La consulta que manda es la primera, la jefa
2. Todas las columnas de cada consulta deben ser del mismo tipo de datos
3. Todas las consultas deben tener el mismo número de columnas

Es unir varias consultas en una SOLA. Es algo conceptual más bien.

En nuestra base de datos actual, tenemos varias tablas que representan personas.

Si nos fijamos en sus datos, no tienen nada que ver entre sí, solo que, para mi, son personas.

Imaginemos que necesitamos, en alguna app, mostrar todos los datos de las Personas de mi BBDD.

Texto

El contenido generado por IA puede ser incorrecto.

Pongamos, sobre esta misma consulta que tenemos, que necesito todas las Personas que cobren más de 250000

Como podemos comprobar, cada consulta es individual, es decir, cada WHERE es propio de cada consulta aunque las tres tengan salario.

Texto

El contenido generado por IA puede ser incorrecto.

Cada consulta es individual.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Además de la clausula UNION, tenemos otro tipo standard.

El operador UNION quita repetidos:

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

### SELECT TO SELECT

Son consultas sobre consultas a su vez, es decir, hacer una consulta sobre un CURSOR.

Es útil cuando queremos ORDENAR un cursor con UNION o hacer un WHERE conjunto sobre un UNION.

Es realizar una consulta, en este ejemplo de la pantalla al campo SUELDO (Por ejemplo)

Tabla

El contenido generado por IA puede ser incorrecto.

Para realizar un SELECT to SELECT necesitamos dos características:

1. Envolver la consulta sobre la que deseemos hacer un select entre paréntesis
2. Que la consulta tenga un ALIAS

Un ALIAS en una consulta se le indica envolviendo la consulta entre paréntesis y, con un espacio después, le indicamos un NOMBRE/ALIAS

Texto

El contenido generado por IA puede ser incorrecto.

## CONSULTAS A NIVEL DE FILA

Este tipo de consultas nos permiten mostrar los datos ya con un formato determinado dentro de la propia consulta.

Nunca cambiamos los datos de la tabla sino que, en el propio select, lo que hacemos es evaluar cada valor y devolver otro valor.

Se utiliza para mostrar los datos directamente para el usuario final sin hacer nada más.

Para visualiza qué significa este tipo de consultas, hacemos una consulta sobre PLANTILLA.

Sintaxis:

Select CAMPO1, CAMPO2

, case CAMPO\_A\_EVALUAR

when VALOR1 then LOQUESEA1

when VALOR2 then LOQUESEA2

else OTROVALOR

end as ALIAS

from TABLA;

Texto

El contenido generado por IA puede ser incorrecto.

# CONSULTAS DE ACCION

En todas las bases de tenemos consultas de acción.

Dichas consultas nos permiten insertar, eliminar o modificar registros.

Oracle realiza las consultas de acción de forma transaccional, lo que quiere decir que no son definitivas en el servidor.

Si elimino una fila, dicha fila es eliminada temporalmente. No veremos el registro, pero no estará borrado.

Si cerramos la sesión, el registro borrado vuelve a aparecer.

Mediante dos palabras podemos indicar si deshacemos los cambios o los mantenemos permanentes cuando estamos con consultas de acción.

1. **PERMANENTE: commit**
2. **DESHACER CAMBIOS: rollback**

Los tipos de consultas de acción son tres:

1. INSERT: Inserta una fila dentro de una tabla de base de datos
2. UPDATE: Modifica uno o varios registros de una tabla
3. DELETE: Elimina uno o varios registros de una tabla

**Nota:** Los where son iguales que en las consultas de selección, es decir, operadores y todo lo que hemos visto.

#### INSERT

Inserta una nueva fila dentro de la tabla.

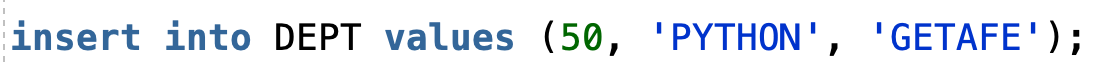
Si deseamos insertar 5 filas, necesitamos 5 insert.

Tenemos dos tipos de sintaxis:

1. **Insertar todos los valores de la tabla**: Esta sintaxis utiliza la tabla como patrón, es decir, los valores que pongamos serán los valores de la tabla en el orden que estén las columnas de la tabla.

Es obligatorio indicar TODAS las columnas de la tabla

**insert into TABLA values (valor1, valor2, valor3, valor4);**

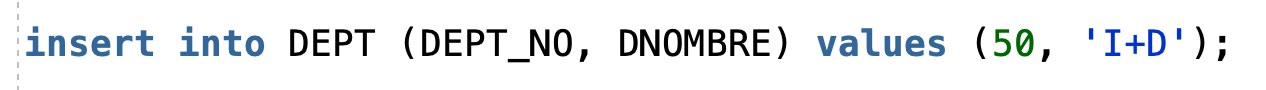


1. **Insertar algunos valores de la tabla, no todos:** Esta sintaxis se utiliza cuando no quiero insertar todos los valores de la tabla para el registro.

Necesito indicar las columnas que voy a utilizar para insertar.

El orden de los valores será el orden de las columnas.

insert into TABLA (COLUMN1, COLUMN2) values (VALOR1, VALOR2);



Dentro de las consultas de acción tenemos algo muy útil que os he dicho que NO deberíamos utilizar, pero en consultas de selección, en consultas de acción es perfectamente válido y muy útil.

Las subconsultas en consultas de selección nos permiten poder recuperar datos de otras tablas o generar elementos dinámicos para las consultas.

Tenemos una nueva persona de la plantilla. Se apellida Lopez y su puesto es ENFERMERA de Tarde en el hospital 22.

Asignar el siguiente **ID de empleado** disponible al insertar el registro.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

insert into PLANTILLA

(EMPLEADO\_NO, APELLIDO, FUNCION, TURNO, HOSPITAL\_COD)

values

((select MAX(EMPLEADO\_NO) + 1 from PLANTILLA), 'Super Lopez', 'ENFERMERA', 'T', 22);

#### DELETE

Este tipo de consulta elimina cero, uno o varios registros a la vez.

Sintaxis:

DELETE FROM TABLA

Aplicación

El contenido generado por IA puede ser incorrecto.

Si la consulta NO tiene un **where**, elimina todos los registros de la tabla.

Es conveniente incluir un **where** para indicar los registros a eliminar

delete from TABLA where CONDICION



También podemos utilizar Subconsultas para nuestro **where**.

Eliminar a las personas de la Plantilla del Hospital **El Carmen**



delete from PLANTILLA where HOSPITAL\_COD=

(select HOSPITAL\_COD from HOSPITAL where NOMBRE='El Carmen');

#### UPDATE

Modifica uno o varios registros de la tabla al mismo tiempo.

En la misma consulta podemos indicar que deseamos modificar más de una columna.

Sintaxis:

update TABLA set COLUMNA1=valor1, COLUMNA2=valor2

Esta consulta anterior, modifica todos los registros de la tabla.

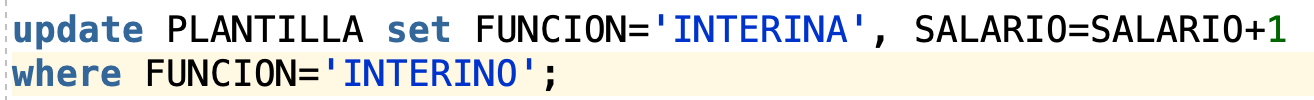
Si lo necesito, podemos incluir un **where**.

update TABLA set COLUMNA1=valor1, COLUMNA2=valor2 where CONDICION

Hoy estamos generosos. Incrementar el salario de la PLANTILLA en 1.



**MODIFICAR LA FUNCION DE LOS INTERINO A INTERINA Y SUBIR EL SALARIO EN 1**

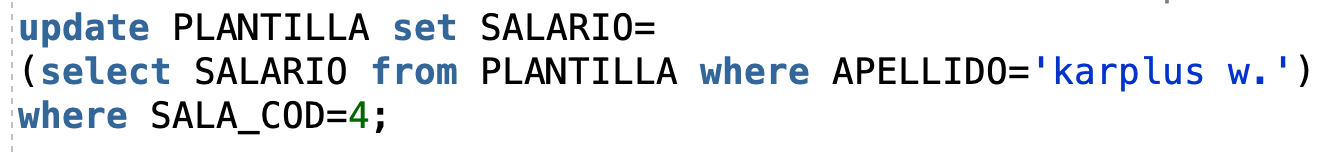


Por supuesto, podemos utilizar subconsultas en UPDATE.

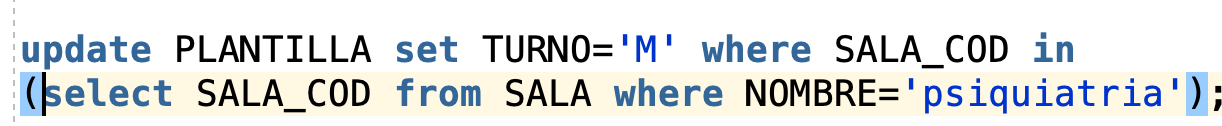
Tanto en el SET como en el WHERE.

Los empleados de la Sala 4 se nos han puesto en Huelga. Dicen que su compi Karplus cobra más que ellos.

Modificar el salario de la Plantilla de la SALA 4 poniendo el mismo salario que KARPLUS



Modificar el TURNO a Mañana a todos los empleados de la Plantilla de la sala de Psiquiatría





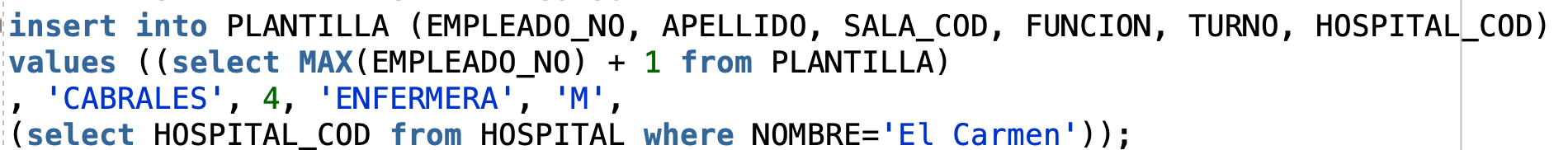
Ejemplo de supuesto:

1) INSERTAR UN NUEVO REGISTRO EN PLANTILLA

LA PERSONA SE LLAMA CABRALES, SALA 4, ENFERMERA, TURNO M

Y TRABAJARA EN EL HOSPITAL DE EL CARMEN. SIN SALARIO.

EL ID SERA EL MAXIMO EN LA CONSULTA



**insert into PLANTILLA (EMPLEADO\_NO, APELLIDO, SALA\_COD, FUNCION, TURNO, HOSPITAL\_COD)**

**values ((select MAX(EMPLEADO\_NO) + 1 from PLANTILLA)**

**, 'CABRALES', 4, 'ENFERMERA', 'M',**

**(select HOSPITAL\_COD from HOSPITAL where NOMBRE='El Carmen'));**

2) ELIMINAR LAS PERSONAS DE LA PLANTILLA

QUE NO TENGAN HOSPITAL

**delete from PLANTILLA where HOSPITAL\_COD not in**

**(select HOSPITAL\_COD from HOSPITAL) or HOSPITAL\_COD is null;**

select \* from PLANTILLA where HOSPITAL\_COD is null;



El último comando (CUIDADO) es la instrucción **TRUNCATE**

La instrucción Truncate es una instrucción que elimina, de una forma rápida, todos los registros de una tabla.

Es rápida porque no tiene ROLLBACK.

En otras bases de datos, hace lo mismo, pero no escribe tampoco en el registro.

Sintaxis: **TRUNCATE TABLE TABLA**

# CONECTAR PYTHON CON ORACLE

Lo que hemos estado realizando son consultas sobre un Servidor de base de datos.

Dichas consultas funcionarán en cualquier servidor de base de datos que tengamos.

Python funciona mediante fabricantes, es decir, si necesitamos conectar con Oracle necesitamos una librería llamada **oracledb** o **cx\_oracle**

Si necesitamos, por ejemplo, consumir SQL Server, utilizaremos una librería llamada **pyodbc**

La mayoría de librerías de acceso a datos son iguales, es decir, todos los objetos son iguales y solamente cambia su cadena de conexión.

Todos tendrán los siguientes elementos:

* CONNECTION: Objeto que nos permite conectar a la base de datos indicando datos como, Servidor, Usuario y Password
* CURSOR: Un objeto cursor es un objeto de base de datos con el resultado de una consulta SELECT.

Cuando ejecutamos cualquier consulta de selección nos devuelve un objeto cursor con las columnas y las filas y que podemos recorrer y recuperar sus datos.

Las librerías se instalan utilizando el comando **pip**

Al estar aquí en el maravillo mundo de macOS, solamente podemos ejecutar pip desde la consola de Visual Studio.

Instalamos la librería que corresponde a Oracle:

**pip install oracledb**



Cada librería depende de una cadena de conexión para acceder al servidor.

Dependiendo del servidor de bases de datos, la cadena cambiará

Para trabajar con Oracle se realiza de la siguiente forma:

1. Importar nuestra nueva librería

import oracledb

1. Mediante un objeto **connection** de la librería, debemos indicar la cadena de conexión.

El servidor de Oracle en su cadena de conexión utiliza **SERVER/BBDD**

connection(user=’USUARIO ORACLE’, password=’PASSWORD’, dsn=’SERVIDOR DE ORACLE’)

Vamos a probar si esto es funcional simplemente creando un simple Python y mostrando un mensaje si conecta.

Creamos un nuevo programa llamado **python33conectaroracle.py**

**#importamos la librería de Python Oracle**

**import oracledb**

**#tenemos un objeto connection(user, password, server)**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("Conectados!!!")**

Sobre el mismo ejemplo, una vez que estamos conectados (siempre debemos hacerlo al acceder a cualquier Server) podríamos realizar una consulta SELECT

Las consultas SELECT se ejecutan mediante un objeto llamado CURSOR y, este objeto, es creado a partir del objeto **connection:**

cursor = connection.cursor()

El cursor, tanto en consultas de selección, como consultas de acción tendrá métodos para ejecutar las consultas:

**cursor.execute(SQL)**

Una vez ejecutada una consulta, en el cursor están los datos.

Podemos recorrer los datos mediante un bucle o fila a fila.

Texto

El contenido generado por IA puede ser incorrecto.

**#importamos la librería de Python Oracle**

**import oracledb**

**#tenemos un objeto connection(user, password, server)**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("Conectados!!!")**

**# LAS CONSULTAS NO LLEVAN ;**

**sql = "select \* from DEPT"**

**# CREAMOS UN CURSOR PARA REALIZAR LA CONSULTA**

**cursor = connection.cursor()**

**#LA CONSULTA SE EJECUTA Y RECUPERA LOS DATOS A LA VEZ**

**#AL APLICAR EL METODO execute(sql)**

**cursor.execute(sql)**

**#UNA VEZ QUE TENEMOS LOS DATOS EN EL CURSOR, DEBEMOS EXTRAERLOS**

**#PARA RECUPERAR LOS DATOS**

**#TENEMOS UN METODO LLAMADO fetchone() QUE NOS DEVUELVE UNA FILA**

**#CADA VEZ QUE EJECUTEMOS fetchone() AVANZA UNA FILA**

**row = cursor.fetchone()**

**#PINTAMOS LA FILA 1**

**print(row)**

**row = cursor.fetchone()**

**#PINTAMOS LA FILA 2**

**print(row)**

**row = cursor.fetchone()**

**#PINTAMOS LA FILA 3**

**print(row)**

**#PINTAMOS LA FILA 4**

**row = cursor.fetchone()**

**print(row)**

**#PINTAMOS LA FILA ????**

**row = cursor.fetchone()**

**print(row)**

**#POR NORMA, SIEMPRE DEBEMOS LIBERAR LOS RECURSOS DE BBDD**

**#CERRAR TODO**

**cursor.close()**

**connection.close()**

**print("Fin de BBDD")**

En el siguiente ejemplo vamos a realizar la misma consulta, pero en lugar de ir fila a fila, recorremos todo a la vez.

Creamos un nuevo programa llamado **python34todosdept.py**

**import oracledb**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**sql = "select \* from DEPT"**

**cursor = connection.cursor()**

**cursor.execute(sql)**

**#PARA RECORRER UN CURSOR SE UTILIZAN BUCLES for**

**for row in cursor:**

**print(row)**

**# SI DESEAMOS EXTRAER DATOS DE ALGUNA COLUMNA**

**# PODEMOS REALIZARLO POR EL INDICE**

**print(row[2])**

**# TAMBIEN PODEMOS RECUPERAR LOS DATOS**

**# POR NOMBRE DE COLUMNA. PERO ESTO**

**# SOLAMENTE ES COMPATIBLE CON ALGUNAS BBDD**

**#nombre = row.DNOMBRE**

**#print(nombre)**

**# EL CURSOR SOLAMENTE SE PUEDE LEER UNA VEZ**

**# SI DESEAMOS LEER DE NUEVO, DEBEMOS EJECUTAR DE NUEVO**

**fila = cursor.fetchone()**

**print(fila)**

**cursor.close()**

**connection.close()**

**print("Fin de BBDD")**

Tenemos también otra posibilidad de recorrer un Cursor.

Podemos hacerlo mediante variables. Una, dos o las que sean dentro del FOR.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

La salida será la siguiente:

Texto

El contenido generado por IA puede ser incorrecto.

En el siguiente ejemplo vamos a probar una consulta para Buscar un departamento por su ID.

Pediremos al usuario el ID del departamento.

Si el departamento no existe, mostraremos un mensaje. Si existe, pintamos los datos.

Hemos visto que si NO encuentra una fila, nos escribe None, pero ese valor NO existe, es solamente cosa de nuestro entorno visual.

Si al leer un registro NO lo encuentra se utiliza el operador **not**

Ejemplo:

if (**not** cursor.fetchone()):

#NO TENEMOS DATOS

if (**not row):**

#NO TENEMOS DATOS

Creamos un nuevo programa llamado **python35buscadordepartamento.py**

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("Conectado a BBDD")**

**print("Introduzca número de departamento")**

**data = input()**

**cursor = connection.cursor()**

**sql = "select \* from DEPT where DEPT\_NO=" + data**

**cursor.execute(sql)**

**#COMO ESTAMOS BUSCANDO POR PK, SOLAMENTE NOS PUEDE**

**#DEVOLVER UNA FILA**

**row = cursor.fetchone()**

**#DEBEMOS COMPROBAR SI FILA TIENE CONTENIDO/ALGO**

**if (not row):**

**print("No existe el departamento")**

**else:**

**#DIBUJAMOS LOS DATOS**

**nombre = row[1]**

**localidad = row[2]**

**print(nombre + ", " + localidad)**

**cursor.close()**

**connection.close()**

**print("Fin de bbdd")**

Creamos un nuevo programa llamado **python36buscadorenfermos.py**

Debemos buscar un enfermo por su **INSCRIPCION.**

Pediremos el dato por pantalla en el programa.

Si el enfermo existe, mostramos su APELLIDO y su DIRECCION

Si no existe, lo indicamos por pantalla.

Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**print("Buscador de Enfermo")**

**print("Introduzca inscripción del Enfermo")**

**data = input()**

**sql = "select \* from ENFERMO where INSCRIPCION=" + data;**

**#CONFIGURAMOS NUESTRAS CONEXIONES**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**cursor = connection.cursor()**

**cursor.execute(sql)**

**row = cursor.fetchone()**

**if (not row):**

**print("No existe el enfermo con la inscripcion " + data)**

**else:**

**apellido = row[1]**

**direccion = row[2]**

**print(apellido + ", Dirección: " + direccion)**

**cursor.close()**

**connection.close()**

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

El siguiente ejemplo que necesitamos hacer es parecido, pero dibujaremos otra consulta

Vamos a realizar una aplicación para buscar empleados de la PLANTILLA por su TURNO.

Pediremos al usuario el turno que desea buscar.

Mostraremos el apellido y la función de los empleados del Turno determinado.

Creamos un programa llamado **python37buscadorplantillaturno.py**

Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**print("Buscador plantilla por turno")**

**print("Introduzca un Turno (T, M, N)")**

**data = input()**

**sql = "select APELLIDO, FUNCION from PLANTILLA where TURNO='" + data + "'"**

**print(sql)**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**cursor = connection.cursor()**

**cursor.execute(sql)**

**#RECORREMOS LOS DATOS DEL CURSOR**

**for apellido, funcion in cursor:**

**print(apellido + ", Función: " + funcion)**

**cursor.close()**

**connection.close()**

**print("Fin de BBDD")**

### CONSULTAS DE ACCION

Las consultas de acción y de selección no se diferencian en nada en el momento de utilizar este lenguaje.

El único cambio será que el cursor ejecuta una consulta de acción, nada más

Al ser una consulta de acción, existe una propiedad llamada **rowcount** que nos devuelve el número de filas que han sido afectadas por la consulta.

Vamos a crear un programa simple donde eliminaremos un enfermo por su inscripción.

Creamos un nuevo programa llamado **python38eliminarenfermo.py**

Texto

El contenido generado por IA puede ser incorrecto.

Como podemos comprobar, no ha eliminado el enfermo dentro de la base de datos, debido a que tenemos que incluir la instrucción final **commit** o **rollback**

Esto es extensible a casi todos los conectores de base de datos, no tiene que ver con la base de datos Oracle.

SQL Server no utiliza ni rollback ni commit, pero si en Python no ponemos esas instrucciones, ocurrirá lo mismo, lo elimina del programa, pero NO de la base de datos.

Las transacciones se realizan mediante el objeto **connection**

**import oracledb**

**print("Eliminar enfermo")**

**print("Introduzca inscripción para borrar")**

**data = input()**

**sql = "delete from ENFERMO where INSCRIPCION=" + data**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**cursor = connection.cursor()**

**#NO EXISTE DIFERENCIA, UN CURSOR EJECUTA CONSULTAS**

**#SI SON DE SELECCION, SE RECORREN**

**#SI SON DE ACCION, RECUPERAMOS SU int MEDIANTE rowcount**

**cursor.execute(sql)**

**afectados = cursor.*rowcount***

**connection.commit()**

**print("Registros eliminados: " + *str*(afectados))**

**cursor.close()**

**connection.close()**

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

Por supuesto, podemos combinar consultas de todo tipo con un solo cursor, no es necesario tener otro cursor abierto si no lo necesitamos.

Por ejemplo:

* Si necesitamos ejecutar dos consultas a la vez, entonces necesitamos dos CURSORES, uno para cada consulta y una sola conexión.
* Si necesitamos ejecutar una consulta después de la otra, solamente necesitamos un único CURSOR.

Para comprobarlo, simplemente vamos a reutilizar el código anterior de Eliminar enfermos y lo que haremos será mostrar todos los enfermos antes/después de eliminar.

**import oracledb**

**print("Eliminar enfermo")**

**print("Introduzca inscripción para borrar")**

**data = input()**

**sql = "delete from ENFERMO where INSCRIPCION=" + data**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**cursor = connection.cursor()**

**#NO EXISTE DIFERENCIA, UN CURSOR EJECUTA CONSULTAS**

**#SI SON DE SELECCION, SE RECORREN**

**#SI SON DE ACCION, RECUPERAMOS SU int MEDIANTE rowcount**

**cursor.execute(sql)**

**afectados = cursor.*rowcount***

**connection.commit()**

**print("Registros eliminados: " + *str*(afectados))**

**cursor.close()**

**#UNA VEZ QUE HEMOS CERRADO EL CURSOR, PODEMOS HACER MAS CONSULTAS**

**#SOBRE EL MISMO OBJETO, CREANDO UNO NUEVO CON LA CONEXION**

**sqlselect = "select \* from ENFERMO"**

**# CREAMOS UN CURSOR SOBRE LA MISMA VARIABLE**

**cursor = connection.cursor()**

**cursor.execute(sqlselect)**

**print("-----ENFERMOS-------")**

**for row in cursor:**

**print("Inscripción: " + *str*(row[0]) + " - " + row[1])**

**#CERRAMOS EL CURSOR**

**cursor.close()**

**connection.close()**

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

Vamos a realizar una aplicación para insertar un nuevo departamento.

Pediremos al usuario los datos del departamento para insertar y mostraremos todos los departamentos al finalizar la instrucción.

Creamos un nuevo programa llamado **python39insertdepartamento.py**

Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**print("Insertar departamentos")**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**#PEDIMOS LOS DATOS AL USUARIO**

**print("Introduzca un Id departamento")**

**iddept = input()**

**print("Introduzca nombre")**

**nombre = input()**

**print("Introduzca localidad")**

**localidad = input()**

**#insert into DEPT values (99,'A','A')**

**sqlinsert = "insert into DEPT values (" + iddept + ",'" + nombre + "','" + localidad + "')"**

**print(sqlinsert)**

**cursor = connection.cursor()**

**cursor.execute(sqlinsert)**

**print("Departamentos insertados: " + *str*(cursor.*rowcount*))**

**cursor.close()**

**cursor = connection.cursor()**

**sqlselect = "select \* from DEPT"**

**cursor.execute(sqlselect)**

**for row in cursor:**

**print(row)**

**cursor.close()**

**connection.close()**

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

Como último ejemplo, quiero realizar lo mismo, es decir, crear un programa para insertar un HOSPITAL.

Pediremos los datos del hospital y mostraremos los hospitales por pantalla al finalizar la inserción.

Creamos un nuevo programa llamado **python40inserthospital.py**

**import oracledb**

**print("Insert hospitales BBDD")**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("Introduzca ID hospital")**

**idhospital = input()**

**print("Nombre de hospital")**

**nombre = input()**

**print("Dirección")**

**direccion = input()**

**print("Teléfono")**

**telefono = input()**

**print("Camas")**

**camas = input()**

**#insert into HOSPITAL values (11,'name','dir','tlf',camas)**

**sqlinsert = "insert into HOSPITAL values (" + idhospital + ",'" + nombre + "','" + direccion + "','" + telefono + "'," + camas + ")"**

**print(sqlinsert)**

**cursor = connection.cursor()**

**cursor.execute(sqlinsert)**

**print("Filas insertadas: " + *str*(cursor.*rowcount*))**

**cursor.close()**

**sqlselect = "select \* from HOSPITAL"**

**cursor = connection.cursor()**

**cursor.execute(sqlselect)**

**for row in cursor:**

**print(row)**

**cursor.close()**

**connection.close()**

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

Una vez que lo hemos visto, la parte de concatenar es muy tediosa, pero si tenemos que hacerlo, pues habrá que hacerlo.

Tenemos una sintaxis, dentro de Python que permite integrar variables dentro de un string y que sean reconocidas, lo que permite poder quitar concatenaciones.

Esta sintaxis no implica conversiones, simplemente concatena y muestra la variable dentro del texto, independientemente a su tipo (int, string)

Lo único que tenemos que hacer es incluir la letra **f** ANTES del string

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**print("Insert hospitales BBDD")**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("Introduzca ID hospital")**

**idhospital = input()**

**print("Nombre de hospital")**

**nombre = input()**

**print("Dirección")**

**direccion = input()**

**print("Teléfono")**

**telefono = input()**

**print("Camas")**

**camas = input()**

**#insert into HOSPITAL values (11,'name','dir','tlf',camas)**

**sqlinsert = *f*"insert into HOSPITAL values ({idhospital},'{nombre}','{direccion}','{telefono}',{camas})"**

**print(sqlinsert)**

**cursor = connection.cursor()**

**cursor.execute(sqlinsert)**

**print(*f*"Filas insertadas: {cursor.*rowcount*}")**

**cursor.close()**

**sqlselect = "select \* from HOSPITAL"**

**cursor = connection.cursor()**

**cursor.execute(sqlselect)**

**for row in cursor:**

**print(row)**

**cursor.close()**

**connection.close()**

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

En el siguiente ejemplo, vamos a realizar lo mismo que hasta ahora, tendremos una consulta de selección y mostraremos los empleados que pertenezcan a un departamento.

Pediremos el número de departamento y mostramos algún dato de los empleados

Creamos un nuevo programa llamado **python41parametros.py**

Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**print("Ejemplo de parámetros Oracle")**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("Introduzca número de departamento")**

**iddept = input()**

**sql = *f*"select \* from EMP where DEPT\_NO={iddept}"**

**print(sql)**

**cursor = connection.cursor()**

**cursor.execute(sql)**

**for row in cursor:**

**print(*f*"Apellido: {row[1]}, Oficio: {row[2]}, Departamento: {row[7]}")**

**cursor.close()**

**connection.close()**

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

Ejecutamos la aplicación y, cuando nos pida el número de Departamento, escribimos lo siguiente:

Texto

El contenido generado por IA puede ser incorrecto.

A esto se le llama SQL Injection.

Mediante consultas SQL, podemos entrar dentro de la base de datos.

En realidad, esta es la consulta que estamos escribiendo:



Cuando estamos concatenando en cualquier programa, lenguaje o plataforma, estamos expuestos a ataques de inyección SQL.

Para prevenir estos ataques se utilizan parámetros en las consultas SQL.

En lugar de concatenar y poner las variables en un string SQL, lo que se hace es incluir “palabras clave” dentro de la consulta y sustituirlas después.

Que se encargue el conector de Oracle y la base de datos.

Cuando utilizamos parámetros, debemos enviar el tipo de dato que necesita la consulta.

Ahora mismo, estamos enviando TEXTO como NUMERO. El DEPT\_NO es un número.



Para enviarlo como parámetro, debe ser un número:

Imagen que contiene Icono

El contenido generado por IA puede ser incorrecto.

La instrucción SQL ya depende de la base de datos.

Por ejemplo, en Oracle, para poder enviar parámetros se realiza con dos puntos antes del nombre del parámetro:



En Oracle los parámetros se incluyen por posición, no por nombre.

En SQL Server, los parámetros se pueden enviar por posición o por nombre:



Una vez que tenemos el parámetro y la consulta, para poder ejecutar la consulta con parámetros se realiza en el método **execute** del cursor



Modificamos el código anterior y realizamos el ejemplo mediante parámetros:

**import oracledb**

**print("Ejemplo de parámetros Oracle")**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("Introduzca número de departamento")**

**iddept = *int*(input())**

**sql = "select \* from EMP where DEPT\_NO=:param1"**

**print(sql)**

**cursor = connection.cursor()**

**cursor.execute(sql, (iddept,))**

**for row in cursor:**

**print(*f*"Apellido: {row[1]}, Oficio: {row[2]}, Departamento: {row[7]}")**

**cursor.close()**

**connection.close()**

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

Realizar un programa en el que incrementaremos el salario de los empleados de un determinado Oficio.

Debemos pedir el OFICIO y el INCREMENTO al usuario.

Mostraremos el número de empleados que han incrementado con un mensaje:

Empleados con incremento: 7

Y mostraremos el APELLIDO, OFICIO y SALARIO de los empleados a los que hemos incrementado el salario.

Creamos un nuevo programa llamado **python42incrementarsalariosemp.py**

Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("Incrementar salario empleados")**

**print("Introduzca incremento")**

**incremento = *int*(input())**

**print("Introduzca el oficio a incrementar:")**

**oficio = input()**

**#CON CONSULTAS SQL SIEMPRE PARAM**

**sqlupdate = "update EMP set SALARIO = SALARIO + :p1 where OFICIO=:p2"**

**cursor = connection.cursor()**

**cursor.execute(sqlupdate, (incremento, oficio))**

**#CONSULTA DE ACCION, DEVUELVE REGISTROS AFECTADOS**

**registros = cursor.*rowcount***

**#ALMACENAMOS EN BBDD**

**connection.commit()**

**#CERRAMOS CURSOR**

**cursor.close()**

**print(*f*"Empleados modificados: {registros}")**

**cursor = connection.cursor()**

**sqlselect = "select APELLIDO, OFICIO, SALARIO from EMP where OFICIO=:p1"**

**cursor.execute(sqlselect,(oficio,))**

**for ape,ofi, sal in cursor:**

**print(*f*"Apellido: {ape}, Oficio: {ofi}, Salario: {sal}")**

**cursor.close()**

**connection.close()**

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

Realizar un ejemplo para incrementar el salario de los empleados de la plantilla de un hospital.

Pediremos el incremento y el código de hospital.

Mostraremos un mensaje con los empleados que han sido modificados: **Modificados: 5**

Después de modificar, mostramos el APELLIDO, FUNCION Y SALARIO de los empleados que hemos modificado

Creamos un nuevo programa llamado **python43incrementarplantilla.py**

Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**print("Incrementar salario Plantilla/Hospital")**

**print("Introduzca ID de hospital")**

**idhospital = *int*(input())**

**print("Introduzca incremento salarial")**

**incremento = *int*(input())**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**sqlupdate = "update PLANTILLA set SALARIO=SALARIO + :p1 where HOSPITAL\_COD=:p2"**

**cursor = connection.cursor()**

**cursor.execute(sqlupdate, (incremento, idhospital))**

**registros = cursor.*rowcount***

**print(*f*"Registros modificados: {registros}")**

**connection.commit()**

**cursor.close()**

**sqlselect = "select APELLIDO, FUNCION, SALARIO from PLANTILLA where HOSPITAL\_COD=:p1"**

**cursor = connection.cursor()**

**cursor.execute(sqlselect, (idhospital,))**

**for ape, fun, sal in cursor:**

**print(*f*"Apellido: {ape}, Función: {fun}, Salario: {sal}")**

**cursor.close()**

**connection.close()**

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

Me gustaría realizar un programa para mostrar los empleados por un determinado Oficio.

El problema está en que no me acuerdo de los oficios que existen…

Para facilitar las cosas a Paco, deberíamos mostrar un “Menú” con los oficios que existan y poder seleccionar un Oficio para mostrar los Empleados.

Creamos un nuevo programa llamado **python44oficiosempleados.py**

Texto, Carta

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("----Buscador Oficios empleados----")**

**sqloficios = "select distinct OFICIO from EMP"**

**cursor = connection.cursor()**

**cursor.execute(sqloficios)**

**contador = 1**

**#NECESITAMOS ALMACENAR EL DATO DE CADA OFICIO**

**listaoficios = []**

**for row in cursor:**

**print(*f*"{contador} - {row[0]}")**

**listaoficios.append(row[0])**

**contador = contador + 1**

**cursor.close()**

**print("Seleccione una opción")**

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

**oficio = listaoficios[opcion - 1]**

**sqlempleados = "select \* from EMP where OFICIO=:p1"**

**cursor = connection.cursor()**

**cursor.execute(sqlempleados, (oficio,))**

**for row in cursor:**

**print(row)**

**cursor.close()**

**connection.close()**

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

Necesitamos un programa para mostrar los empleados de la plantilla de un determinado TURNO.

Debemos seleccionar un Turno de una serie de opciones y mostraremos los datos de la plantilla de dicho Turno.

Creamos un nuevo programa llamado **python45turnoplantilla.py**

Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("-------Plantilla por turnos-------")**

**sqlturnos = """**

**select distinct TURNO, case TURNO**

**when 'T' then 'TARDE'**

**when 'M' then 'MAÑANA'**

**else 'NOCHE'**

**end as VALOR**

**from PLANTILLA**

**"""**

**cursor = connection.cursor()**

**cursor.execute(sqlturnos)**

**listaTurnos = []**

**contador = 1**

**for row in cursor:**

**listaTurnos.append(row[0])**

**print(*f*"{contador} .- {row[1]}")**

**contador += 1**

**cursor.close()**

**print("Seleccione una opción")**

**opcion = *int*(input()) - 1**

**turno = listaTurnos[opcion]**

**sqlplantilla = "select \* from PLANTILLA where TURNO=:p1"**

**cursor = connection.cursor()**

**cursor.execute(sqlplantilla, (turno,))**

**for row in cursor:**

**print(row)**

**cursor.close()**

**connection.close()**

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

Queremos un programa que nos permita insertar un Doctor en un Hospital determinado.

El programa tendrá que ayudarnos para indicarnos en qué Hospital vamos a insertar al Doctor

Dentro de nuestro programa debemos pedir los siguientes datos:

1. Apellido del Doctor
2. Especialidad
3. Salario

Mostraremos un menú para que seleccionemos el Hospital que deseemos.

1.- La paz

2.- General

3.- El Carmen

Insertaremos el doctor en dicho Hospital

El ID del doctor lo debemos recuperar de la base de datos como el máximo ID del doctor (Otra consulta)

**python46insertdoctorhospital.py**

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**import oracledb**

**connection = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

**print("------INSERT DOCTOR HOSPITAL-----")**

**print("Introduzca un Apellido")**

**apellido = input()**

**print("Especialidad del doctor")**

**espe = input()**

**print("Salario del doctor")**

**salario = *int*(input())**

**sqlhospitales = "select \* from HOSPITAL"**

**cursor = connection.cursor()**

**cursor.execute(sqlhospitales)**

**contador = 1**

**listaHospitales = []**

**for row in cursor:**

**listaHospitales.append(row[0])**

**print(*f*"{contador}) {row[1]}")**

**contador += 1**

**cursor.close()**

**print("Seleccione un hospital para el Doctor")**

**opcion = *int*(input()) - 1**

**idhospital = listaHospitales[opcion]**

**sqlmaxId = "select MAX(DOCTOR\_NO) + 1 as MAXIMO from DOCTOR"**

**cursor = connection.cursor()**

**cursor.execute(sqlmaxId)**

**row = cursor.fetchone()**

**iddoctor = row[0]**

**cursor.close()**

**sqlinsert = """**

**insert into DOCTOR values (:idhospital, :iddoctor**

**, :apellido, :espe, :salario)**

**"""**

**cursor = connection.cursor()**

**cursor.execute(sqlinsert, (idhospital,iddoctor,apellido, espe, salario))**

**registros = cursor.*rowcount***

**print(*f*"Doctores insertados: {registros}")**

**connection.commit()**

**cursor.close()**

**connection.close()**

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

El siguiente paso que vamos a realizar será separar el acceso a datos de la lógica que estamos realizando.

Por un lado estarán los Print y demás elementos como los input()

Por otro lado estará la capa de acceso a datos, que tendrá los cursores y las consultas

import oracledb

class ClaseConexionOracle:

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

connection = Oracledb.connect(…..)

def insertarAlgo(self, valor1, valor2):

sqlinsert = “insert into TABLA values (valor1..)”

cursor = self.connection.cursor()

**PYTHON PROGRAM**

import ClaseConexionOracle;

claseDatos = ClaseConexionOracle()

print(“CCCC”)

print(“Dame datos”)

valor1 = input()

claseDatos.insertAlgo(valor1);

Vamos a crear una lógica ya realizada, el ejemplo de Eliminar Enfermos por su inscripción.

La eliminación, conexión y demás características las haremos en una Clase de datos.

Pedir la inscripción y la llamada a eliminar, la haremos en una clase de Python.

A nuestras clases de datos las vamos a llamar **conexionoracle47eliminarenfermos.py**

**CONEXIONORACLE47ELIMINARENFERMOS.PY**

**import oracledb**

***class* ConexionOracleEnfermos:**

**#EN EL INICIO DE LA CLASE DEBEMOS CREAR**

**#UN OBJETO connection PARA UTILIZARLO EN**

**#TODOS LOS METODOS (\_\_init\_\_)**

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

***self*.*connection* = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

***def* eliminarEnfermo(*self*, *inscripcion*):**

**cursor = *self*.*connection*.cursor()**

**sql = "delete from ENFERMO where INSCRIPCION=:p1"**

**cursor.execute(sql, (inscripcion,))**

**registros = cursor.*rowcount***

**cursor.close()**

**return registros**

A continuación, creamos una clase para llamar al método de eliminar.

**PYTHON47ORACLEELIMINARENFERMO.PY**

**from conexionoracle47eliminarenfermos import ConexionOracleEnfermos**

**print("Probando clases de Oracle: ENFERMOS")**

**print("Introduzca una inscripción: Eliminar")**

**inscripcion = *int*(input())**

**#CREAMOS UN OBJETO CONNECTION PARA EJECUTAR LAS CONSULTAS**

**connection = ConexionOracleEnfermos()**

**eliminados = connection.eliminarEnfermo(inscripcion)**

**print(*f*"Enfermos eliminados: {eliminados}")**

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

Quiero un método que pueda modificar el apellido de un Enfermo por su inscripción.

Haremos el método dentro de la clase **conexión**

En la clase **Python** creamos un menú que pregunte si queremos Eliminar o Modificar y hacemos una acción u otra.

**CONEXION**

**import oracledb**

***class* ConexionOracleEnfermos:**

**#EN EL INICIO DE LA CLASE DEBEMOS CREAR**

**#UN OBJETO connection PARA UTILIZARLO EN**

**#TODOS LOS METODOS (\_\_init\_\_)**

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

***self*.*connection* = oracledb.connect(*user*='SYSTEM', *password*='oracle', *dsn*='localhost/xe')**

***def* eliminarEnfermo(*self*, *inscripcion*):**

**cursor = *self*.*connection*.cursor()**

**sql = "delete from ENFERMO where INSCRIPCION=:p1"**

**cursor.execute(sql, (inscripcion,))**

**registros = cursor.*rowcount***

**cursor.close()**

**return registros**

***def* modificarEnfermo(*self*, *apellido*, *inscripcion*):**

**sql = "update ENFERMO set APELLIDO=:p1 where INSCRIPCION=:p2"**

**cursor = *self*.*connection*.cursor()**

**cursor.execute(sql, (apellido, inscripcion))**

**registros = cursor.*rowcount***

**cursor.close()**

**return registros**

# ARQUITECTURA DE PROYECTOS

Los proyectos suelen tener separadas las diferentes capas de forma que, podamos modificar una capa por otra sin casi modificar nada.

En donde estamos, vamos a visualizar capas de Acceso a datos.

No importa que estemos con Oracle o SQL, es lo de menos, es una capa de servicios que nos ofrecerá lo que necesitemos para nuestro dibujo.

Debemos de aislarnos del proveedor de datos, es decir, no utilizar sus características en los dibujos, solamente en los servicios.

El sistema de la arquitectura es trabajar de la siguiente forma:

* **Models:** Son las clases que representan nuestros datos: Empleado, Departamento…
* **Services:** Son las clases que nos proporcionan los datos/modelos, desde Oracle o SQL Server, por ejemplo.
* **Main:** Son las clases que tendrán los dibujos, ya sean clases Print, clases Web o clases con dibujos de escritorio.

Trabajaremos con carpetas dentro de nuestro proyecto para poder organizar todo de una manera más visual.

Al trabajar con subcarpetas dentro de Python, debemos crear un paquete con las clases que tenga dicha carpeta para poder llamar a las clases mediante **import**

Para crear un paquete dentro de una carpeta debemos agregar un fichero vacío llamado **\_\_init\_\_.py**

El nombre del paquete será el nombre de la CARPETA

Vamos a trabajar con un nuevo proyecto.

Lo que voy a hacer es copiar este documento al otro proyecto para seguir actualizando.

Comenzamos creando un nuevo proyecto llamado **PythonDatos**

Creamos la siguiente estructura de carpetas:

PythonDatos

services

models

main01.py

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Vamos a crear un **Servicio** de prueba para utilizarlo en MAIN

Sobre **services** creamos un nuevo Python **service01prueba.py**

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

El contenido generado por IA puede ser incorrecto.

Vamos a intentar hacer un **import** desde el **main**

Como vemos, no aparecen el método getSaludo()

Si queremos utilizarlo, necesitamos un paquete de la carpeta **services**

Para generar un paquete, necesitamos tener un fichero, dentro de la carpeta services, llamado

**\_\_init\_\_.py**

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.

A continuación, vamos a tener un Modelo, es decir, una clase que tendrá una serie de propiedades y que nos la enviará el Servicio hasta el MAIN

Sobre **models** creamos un nuevo fichero llamado **\_\_init\_\_.py** y lo dejamos vacío

Sobre **models** creamos una nueva clase llamada **mascota.py**

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

El contenido generado por IA puede ser incorrecto.

El servicio será el encargado de darnos una mascota.

Abrimos **service01prueba.py** y creamos un nuevo método

**from models import mascota**

**#ESTE SERVICIO SERA EL QUE TENDRA METODOS**

**#PARA SER UTILIZADOS EN EL MAIN**

***def* getSaludo():**

**return "Bienvenido a Matrix"**

***def* getMascota1():**

**dato = mascota.Mascota()**

**dato.*nombre* = "Flounder"**

**dato.*raza* = "Pez"**

**dato.*edad* = 22**

**return dato**

***def* getMascota2():**

**dato = mascota.Mascota()**

**dato.*nombre* = "Nala"**

**dato.*raza* = "Leona"**

**dato.*edad* = 17**

**return dato**

Las clases solamente se pueden ejecutar desde los **MAIN**

Abrimos **main01.py**

**from services import service01prueba**

**from models import mascota**

**saludo = service01prueba.getSaludo()**

**print("Todo OK, " + saludo)**

**pez = service01prueba.getMascota1()**

**leona = service01prueba.getMascota2()**

**print(pez.*nombre*)**

**print(leona.*nombre*)**

Tenemos la posibilidad de incluir ALIAS a nuestros import, de forma que se llamarán con dicho alias a lo largo de nuestra aplicación.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

El siguiente concepto está en quitar y poner piezas para cambiar los datos que vienen de nuestro Servicio.

Para probarlo, vamos a crear otro servicio distinto (Con los mismo métodos) y quitar una pieza por otra en el main y veremos que todo sigue igual.

Sobre **services** creamos un nuevo servicio llamado **service02prueba.py**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.