PYTHON

El software a instalar:

\* Cuenta de Github: [www.github.com](http://www.github.com)

\* Python: Compilador

<https://www.python.org>

\* Visual Studio Code: IDE que es el entorno de trabajo para hacer programas con diversos lenguajes

<https://code.visualstudio.com>

La configuración VS Code solamente la vamos a realizar una vez.

En casa, tendremos la misma configuración: Colores y demás.

Vamos a sincronizar con nuestra cuenta de Github

En Casa:

Descargar Git for Windows y lo instalamos

<https://git-scm.com/install/windows>

Abrimos la consola de comandos:

Escribimos en el buscador: **CMD**

Configuramos el User name de Github

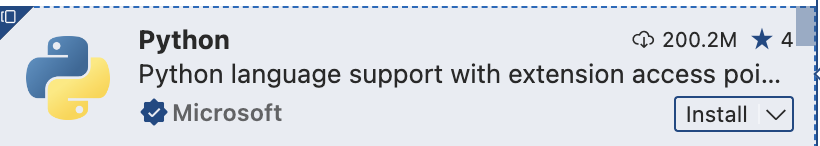
**git config --global user.name "usuario github"**

Configuramos el email de github

**git config --global user.email pacoserranox@gmail.com**



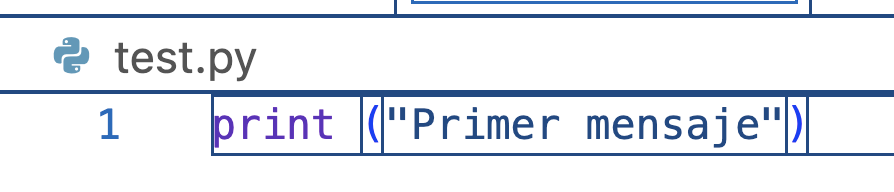
Abrimos VS Code e instalamos nuestras extensiones:



Comprobar si esta bien instalado

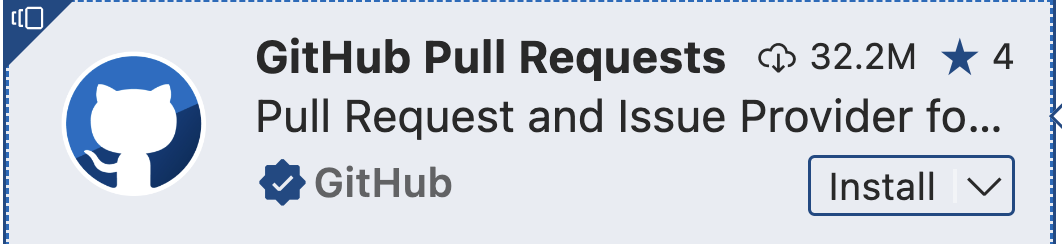
Vamos a crear una carpeta llamada test en Documents

Abrimos la carpeta con VS Code y creamos un nuevo fichero llamado **test.py**

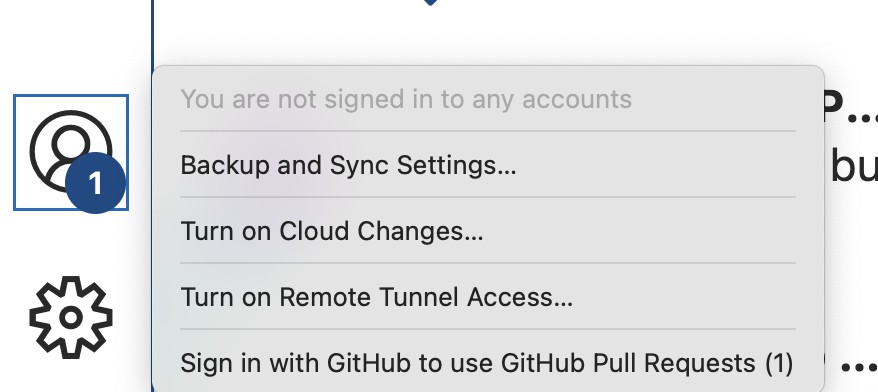


Debemos ejecutar este fichero: F5 o el botón de PLAY

Queremos sincronizar nuestros proyectos con Casa y a la inversa también

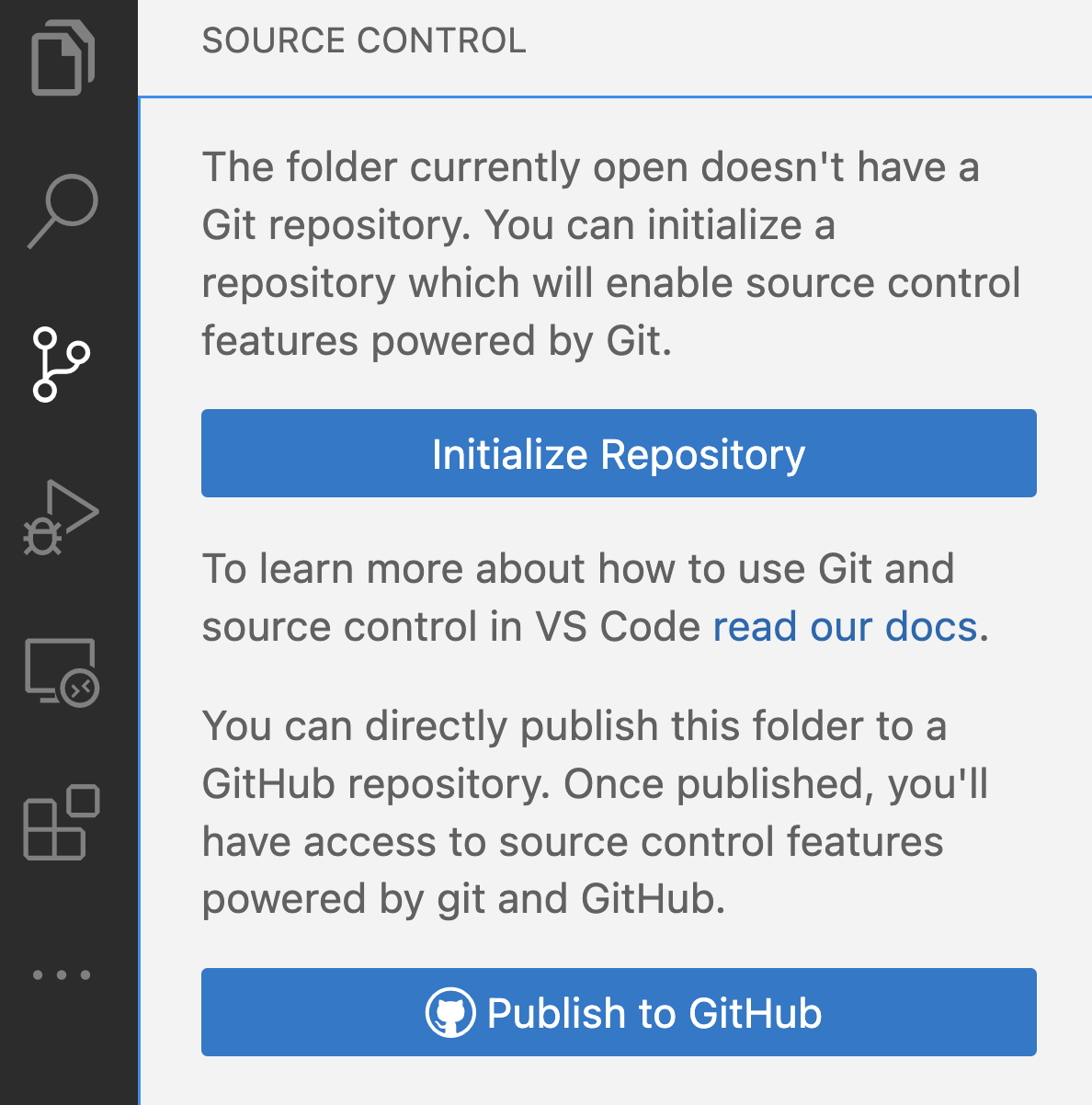


Una vez instalado, debemos validarnos con nuestra cuenta de Github para sincronizar todo.

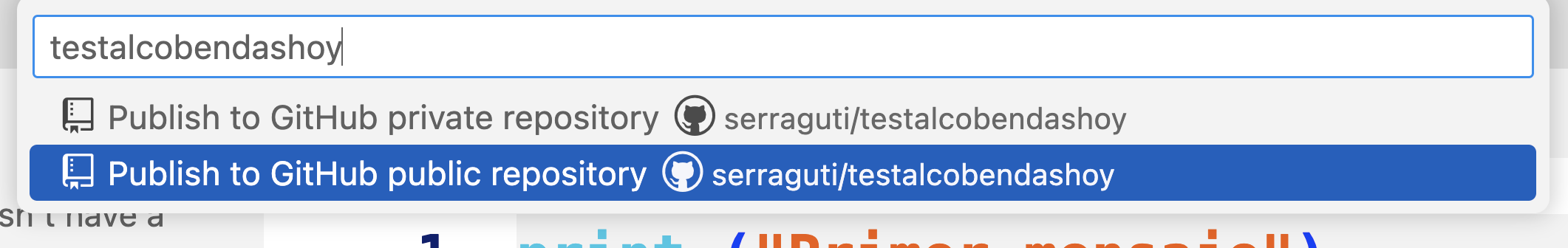


Extensiones: Python, Git

Subimos nuestro primer proyecto de prueba a Github



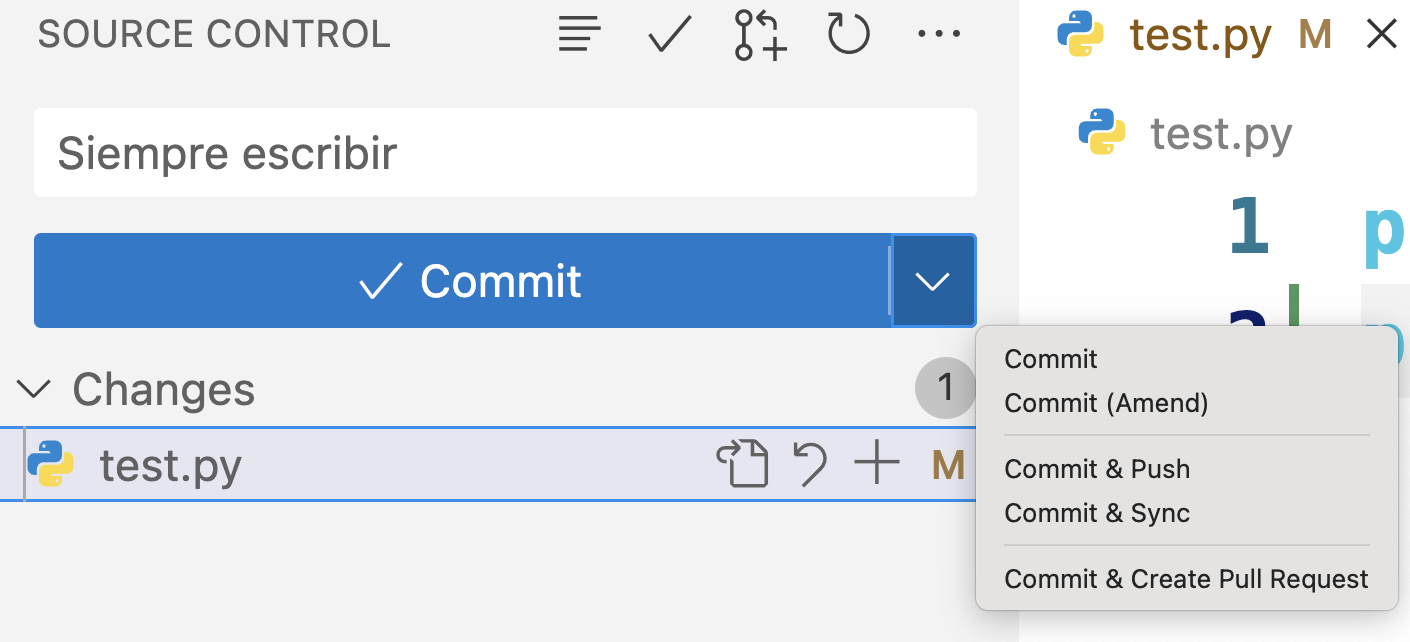
Pulsamos en **Publish to …**



Una vez publicado, ya podremos subir los cambios de nuestro proyecto/carpeta.

Para subir cuando tenemos cambios en nuestro código:

Pulsamos en Commit and Push



Entender el funcionamiento de Git

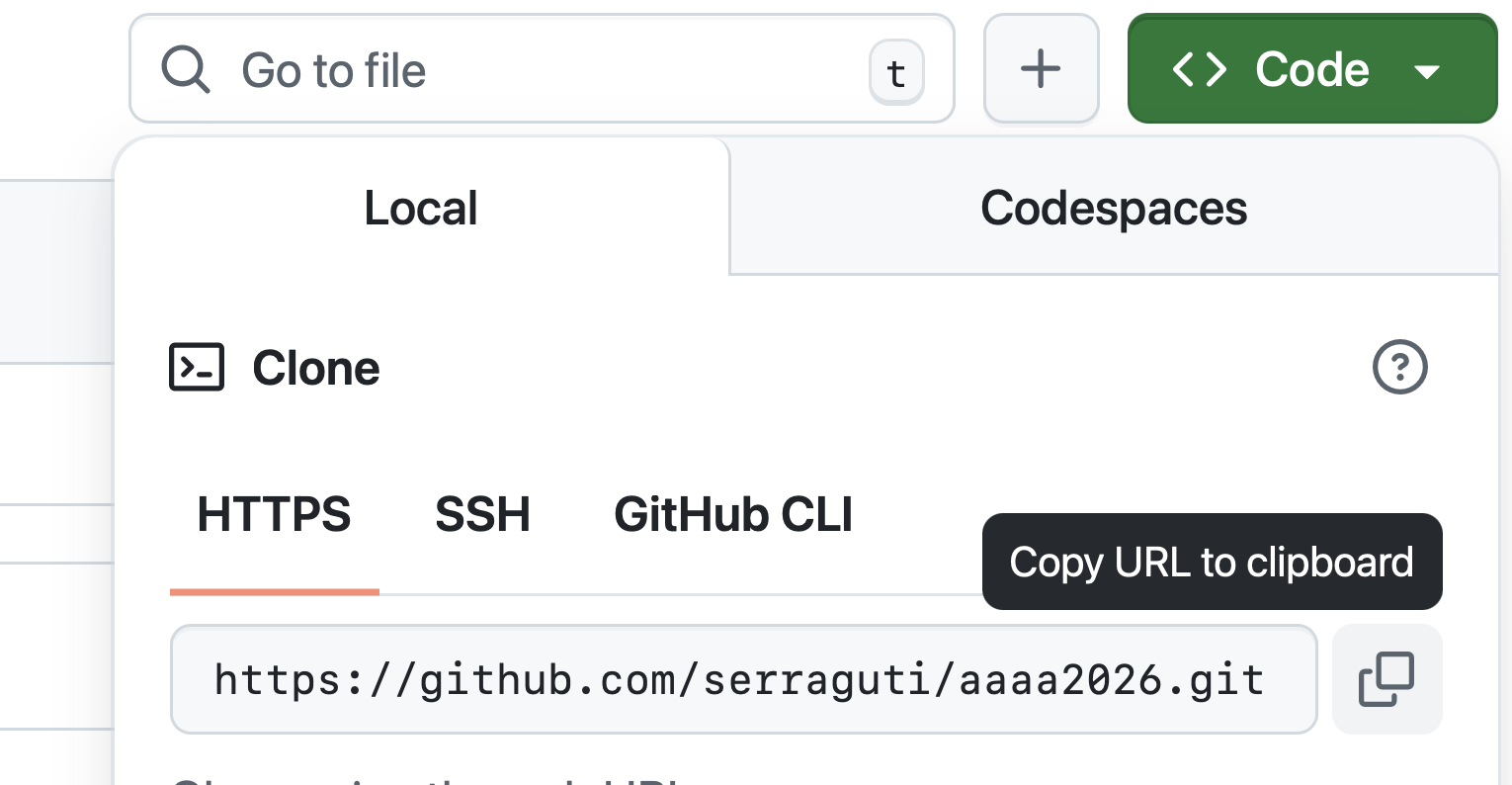
1) Subir un proyecto con VS Code

2) Realizar cambios en el proyecto

3) Subir los cambios y sincronizarlos

El siguiente paso es visualizar cómo trabajo en casa con este proyecto.

1) Tendremos que acceder a nuestro proyecto de Github y copiar la dirección **.git**



2) Abrir Visual Studio e indicar **Clone Repository**

3) Indicar dónde queremos guardar el proyecto en nuestro equipo

4) Podemos hacer cambios y subirlos sin problemas

**Sincronizar con casa y el aula**

En casa, he realizado un cambio y lo he subido a Github.

En el aula, no tengo dicho cambio.

Para sincronizar, primero en el aula, tengo que hacer un PULL

**Si nos cargamos nuestro proyecto en Local, podemos subir una nueva copia**

Dentro de la carpeta de nuestro proyecto, vemos los ficheros ocultos y eliminamos **.git**

Command + Shift + .

LENGUAJE PYTHON

Es un lenguaje compilado, debemos ejecutar cada fichero para probarlo.

La extensión de los ficheros de ejecución es **.py**

Diferencia mayúsculas de minúsculas.

Importante los espacios. No debemos dejar espacios/tabuladores en las líneas seguidas de código.

Vamos a trabajar de la siguiente forma:

Tendremos una carpeta llamada **fundamentospython**

En esta unidad, tendremos aquí todos los programas que vayamos viendo en la parte de fundamentos.

Los ficheros vamos a nombrarlos de una forma para tener una sincronización de números.

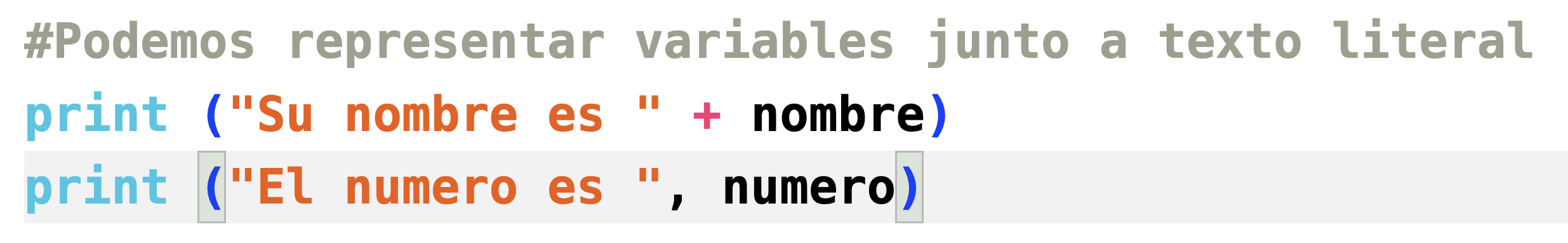
Los nombres de los ficheros no comenzaran por número.

Los iremos nombrando **python01.py**

Creamos un nuevo programa **python01variables.py**

Las variables NO tienen tipado en Python. Cuando declaramos una variable, no le decimos el tipo de dato que almacenaremos.

En otros lenguajes, estoy obligado a indicar el tipo.



Las variables tienen un tipado que se establece en el momento de asignar un valor.

Dependiendo del tipo, podrán realizar unas acciones u otras.

Por ejemplo, una variable de tipo número (int) puede realizar operaciones matemáticas.

Operadores matemáticos:

Suma: +

Resta: -

División: /

Multiplicación: \*

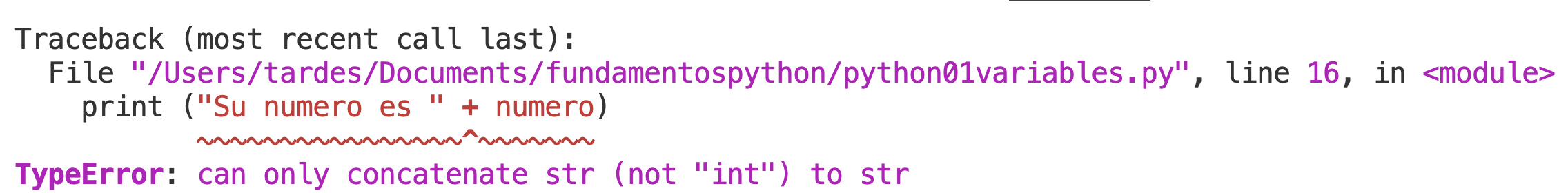
Tenemos tres tipos de errores en cualquier código:

1) Errores de compilación. Estamos escribiendo mal. El compilador (IDE) nos dice que tenemos algo mal escrito



2) Errores en ejecucion. Todo está bien escrito, pero no hemos tenido en cuenta algo de nuestro código y nos falla al ejecutar.





3) Errores lógicos. Todo está bien escrito, todo funciona, no nos da ningún error, pero el programa no hace lo que queremos. El problema soy yo.

Dentro de Python, tenemos funciones de conversión entre tipos de dato.

Por ejemplo, podemos convertir un número a texto

Podemos convertir un texto a número.

Para convertir entre tipos tenemos estas funciones:

\* str(valor): Convierte un valor a string.

\* int(string): Convierte un texto a int

\* float(string): Convierte un texto a decimal

**Nota: Las funciones de conversión no hacen milagros.**





**PYTHON01VARIABLES.PY**

**#Comentarios en Python**

**#Option derecha y 3**

**#Dibujar por pantalla es con string/texto**

**print ("Mi primer Python")**

**#Cada instruccion en una linea**

**#Declara variables: simplemente nos inventamos un nombre**

**nombre = "Paco"**

**numero = 14**

**numero2 = 6**

**suma = numero + numero2**

**resta = numero - numero2**

**division = numero / numero2**

**multi = numero \* numero2**

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

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

**print("Division ", division)**

**print("Multiplicacion ", multi)**

**print (nombre)**

**print (numero)**

**#Podemos representar variables junto a texto literal**

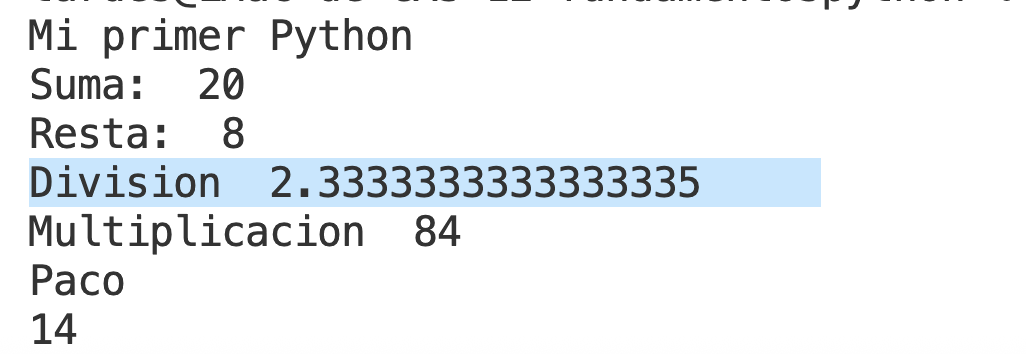
**print ("Su nombre es " + nombre)**

**print ("El numero es ", numero)**

**#Las variables tienen un tipo de dato interno**

**#Si intentamos concatenar un texto con un numero con +**

**print ("Su numero es " + *str*(numero))**



**PEDIR INFORMACION AL USUARIO**

Lo que acabamos de realizar es la utilización de valores estáticos dentro de las variables.

El siguiente paso es preguntar al usuario y que nos ofrezca algún valor para nuestro programa.

Para pedir valores se utiliza la función **input()**

Todos los valores son STRING aunque pidamos un número, por lo que debemos utilizar funciones de conversión.

Vamos a crear un programa para visualizar cómo pedir datos

Creamos un nuevo programa llamado **python02usuario.py**

**print("Ejemplo información del usuario")**

**#Preguntar al usuario algo**

**print("Dame un numero")**

**#Almacenar el dato que nos ha dado en una variable**

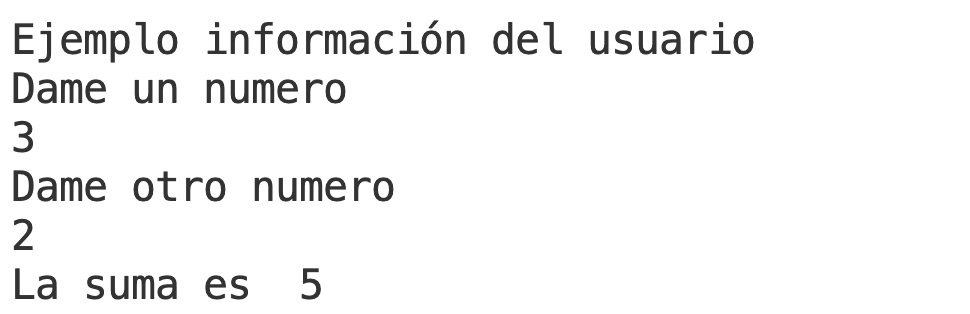
**numero1 = input()**

**print("Dame otro numero")**

**numero2 = input()**

**suma = *int*(numero1) + *int*(numero2)**

**print("La suma es ", suma)**



**CONDICIONALES**

Ahora mismo, nuestro código es lineal, siempre hace lo mismo.

Un condicional permite que un programa pueda cambiar el funcionamiento dependiendo de valores en el programa.

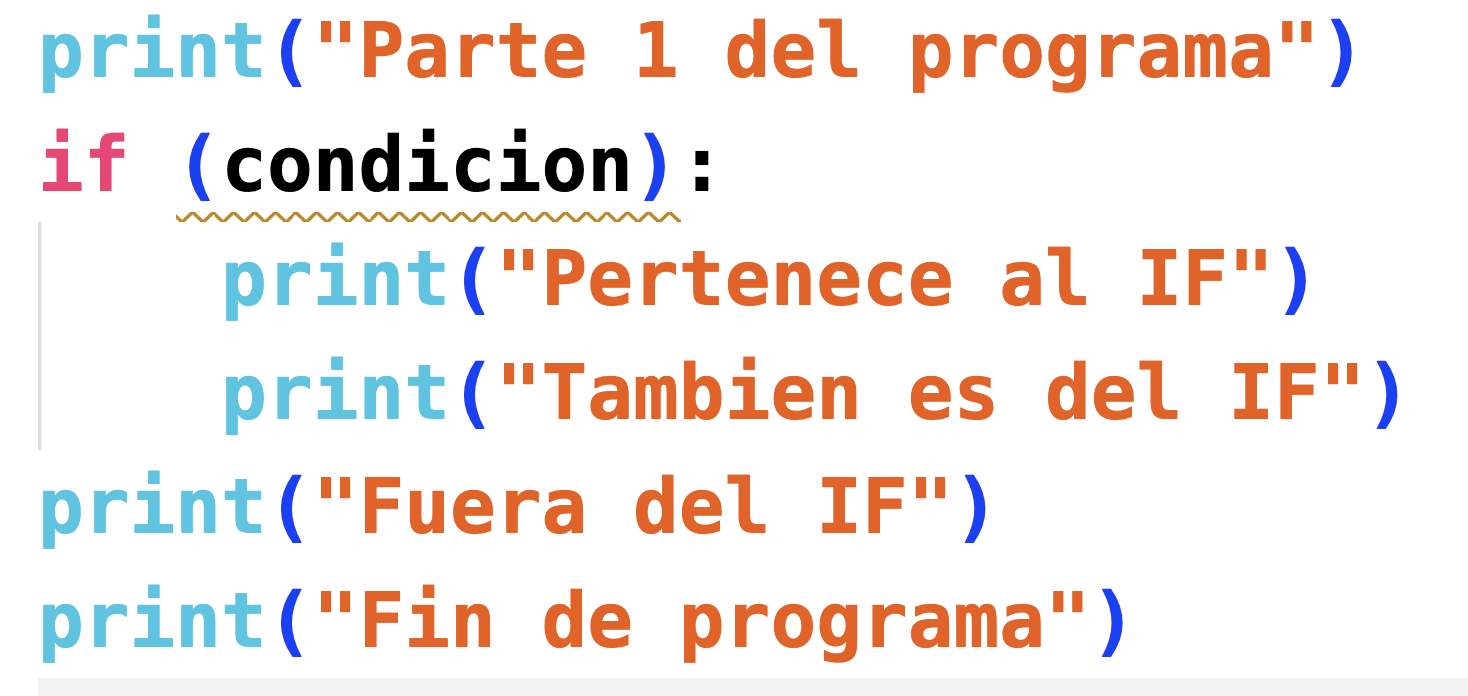
No deja de ser hacer preguntas en el programa.

Los condicionales se declaran mediante una palabra llamada **if**

Los condicionales se establecen a partir de tabulaciones.

Su código es ejecutado si TABULAMOS

Todo lo que esté tabulado dentro del código pertenece a una pregunta if.



Para poder hacer preguntas necesitamos los operadores condicionales.

> Mayor

>= Mayor o igual

< Menor

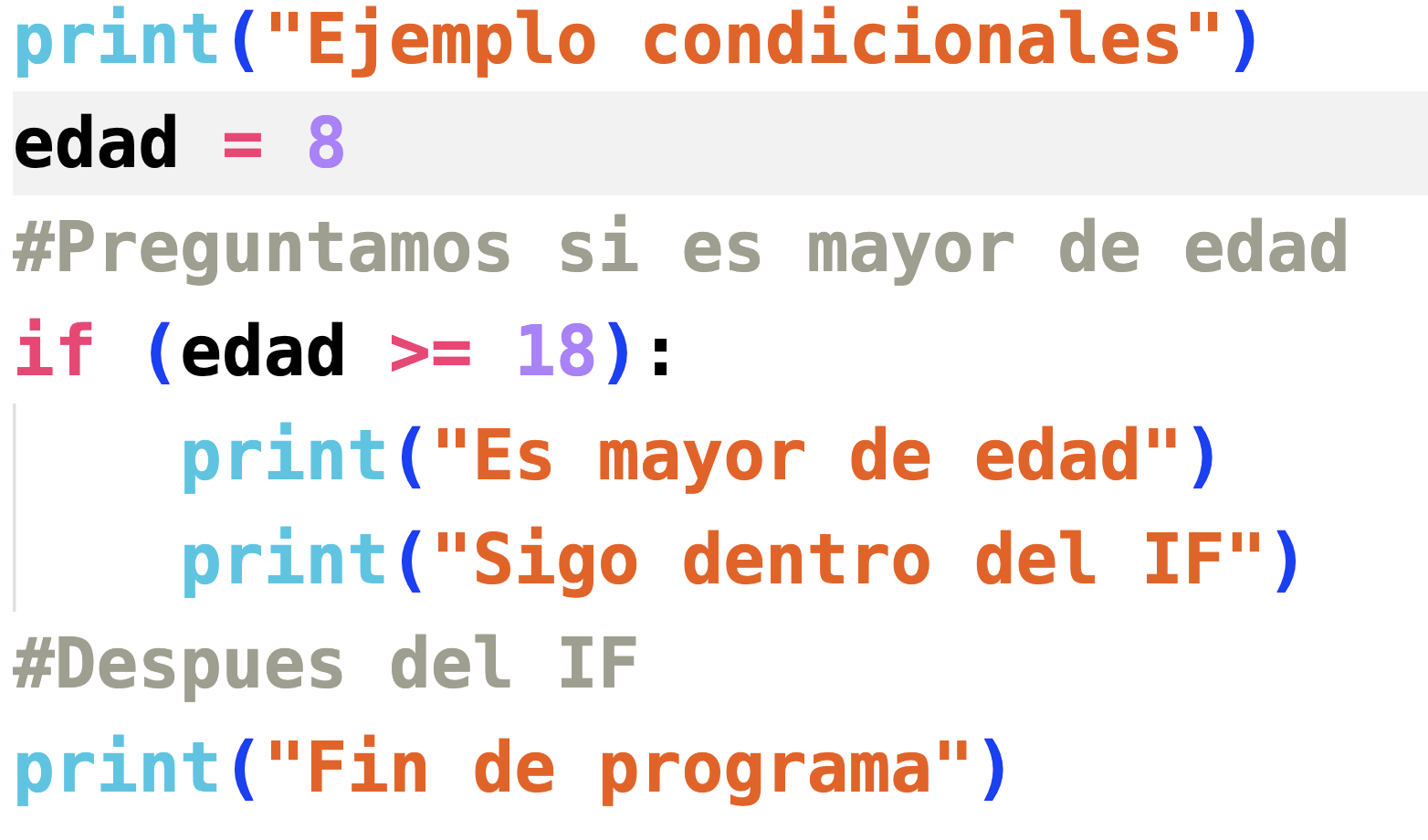
<= Menor o igual

== Igual

!= Distinto

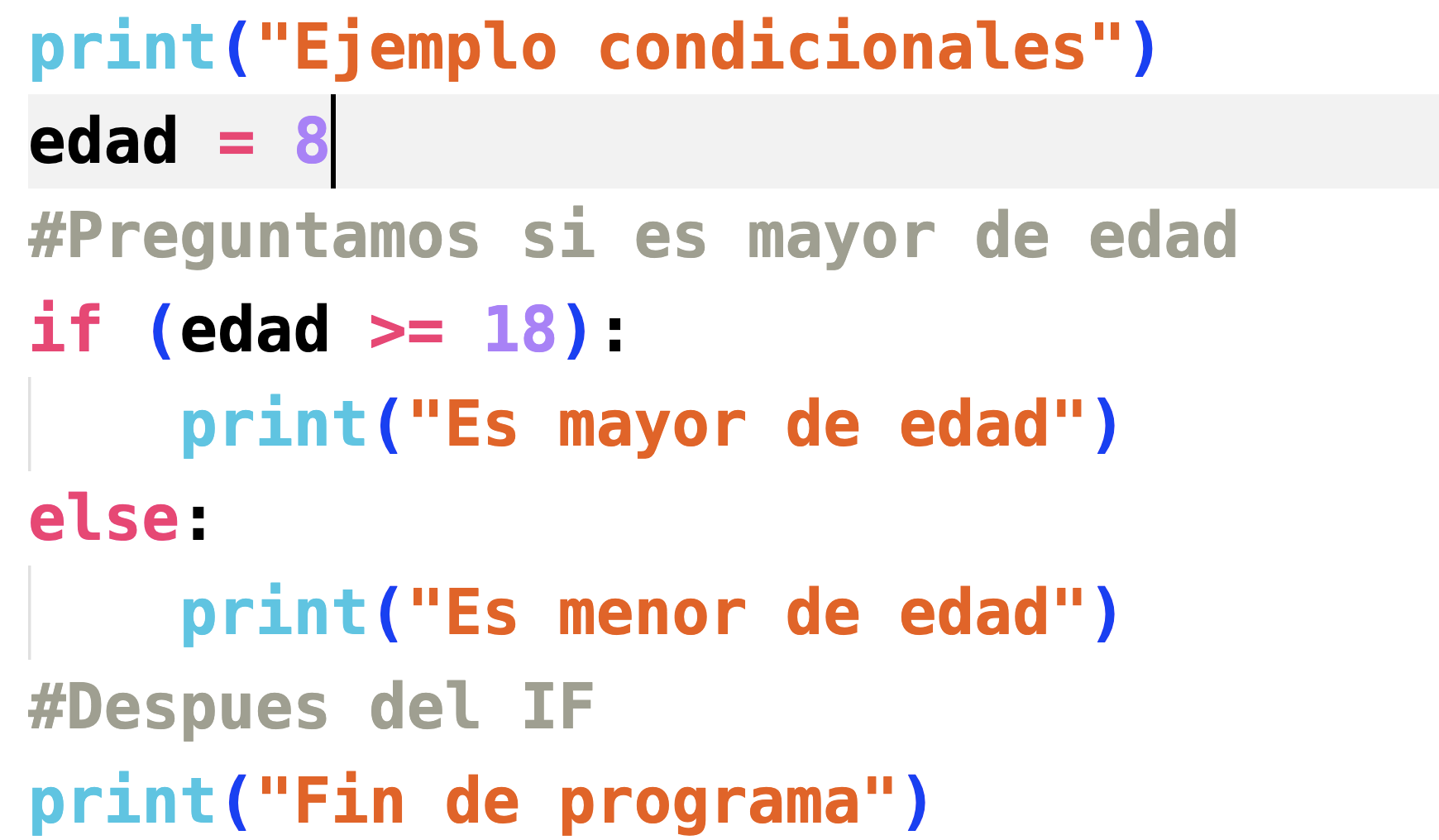
Creamos un nuevo programa llamado

**python03condicionales.py**



Puede ser que deseemos mostrar un mensaje si se cumple la condición (es mayor de edad) y otro mensaje si no se cumple la condición

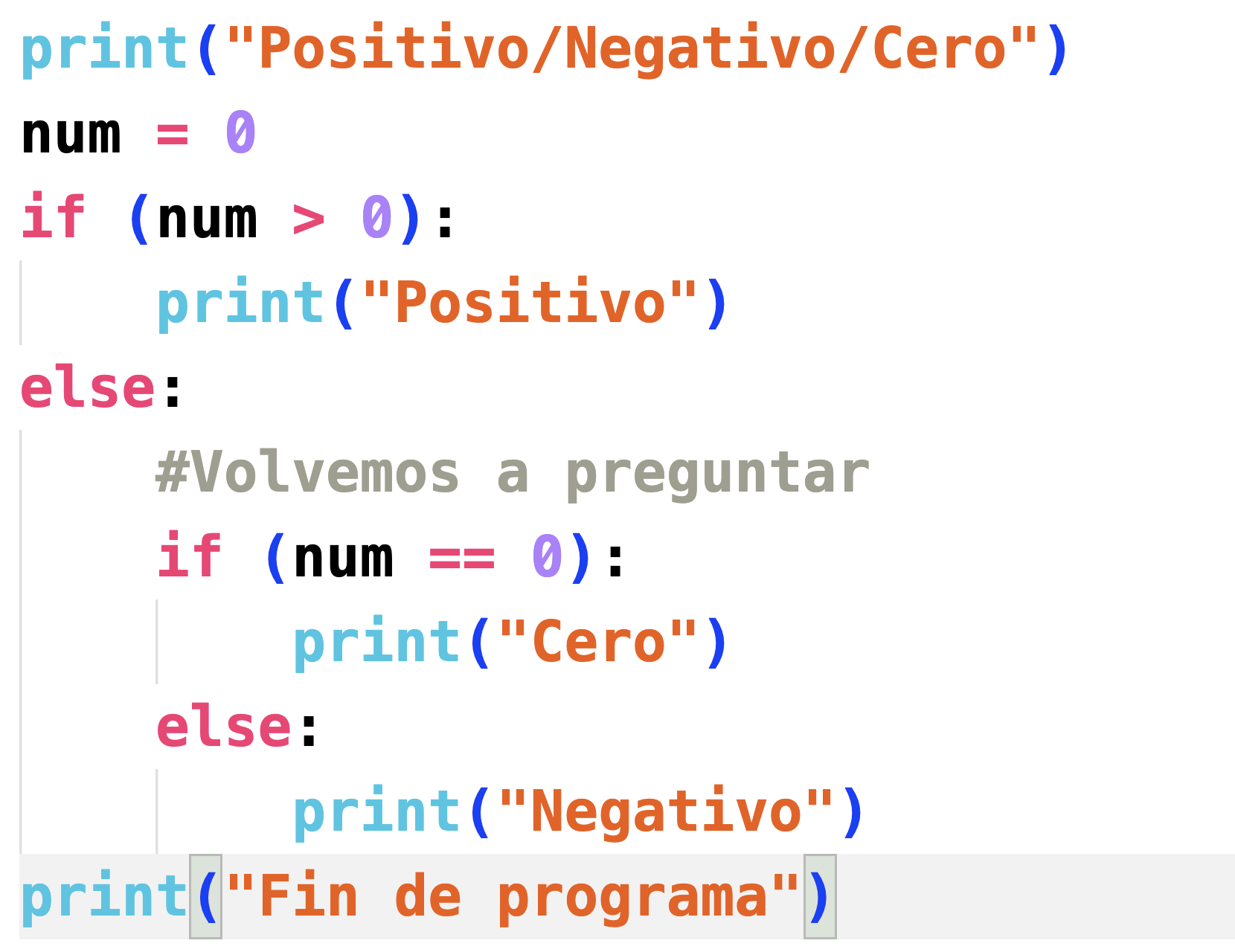
El tipo de if es **if..else**



Tenemos una tercera opción dentro de los if, que es hacer más preguntas sobre una misma cuestión

Para probarlo vamos a crear otro programa llamado

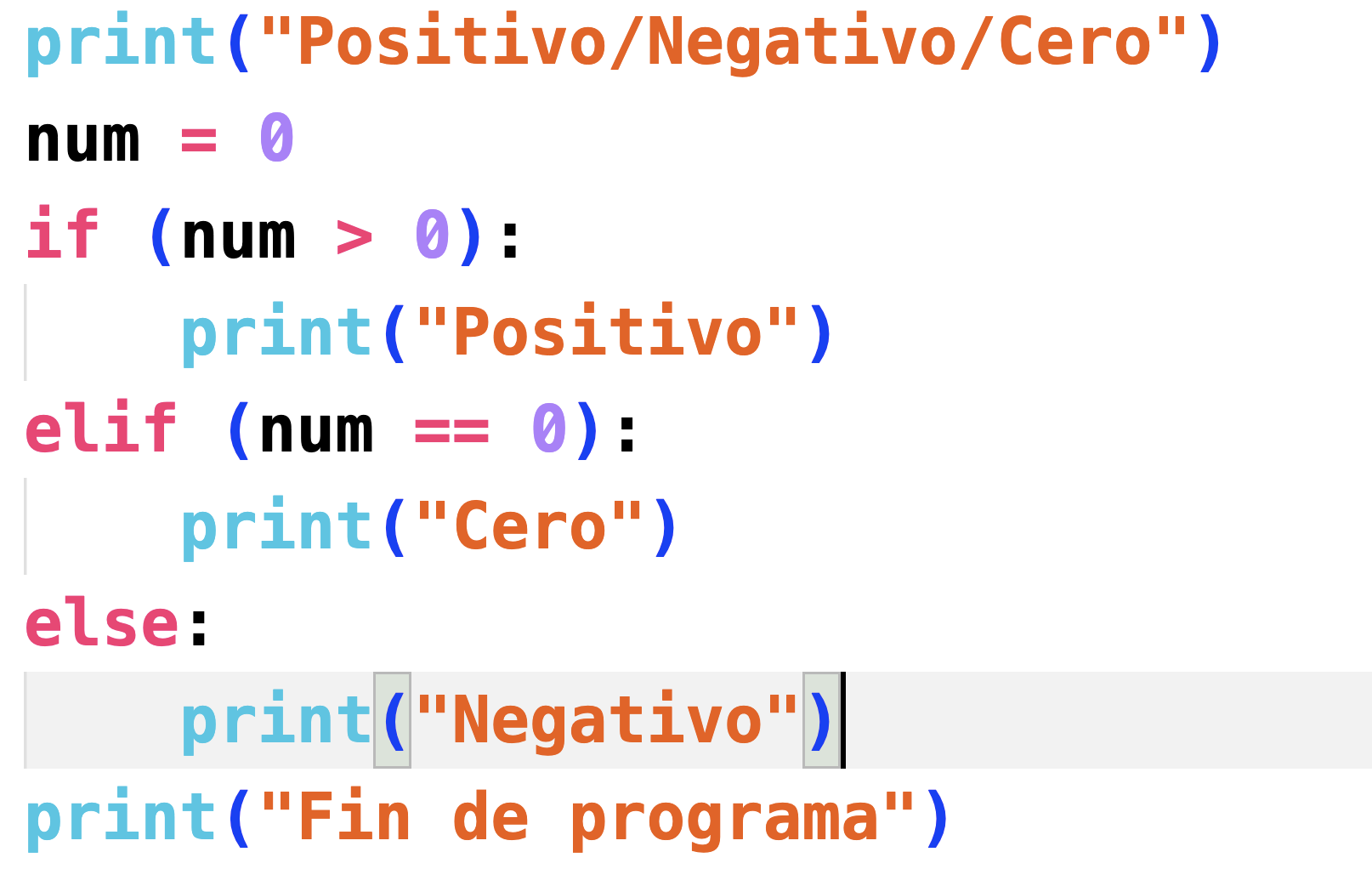
**python04positivonegativo.py**



Tenemos otro tipo de sintaxis del if para cuando queremos preguntar sobre una misma variable.

Dicha sintaxis es llamada **if..else if..else**

Modificamos el código anterior y lo hacemos con esa sintaxis



Por último, podemos hacer dinámico el programa pidiendo el número al usuario.

**print("Positivo/Negativo/Cero")**

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

**num = *int*(input())**

**if (num > 0):**

**print("Positivo")**

**elif (num == 0):**

**print("Cero")**

**else:**

**print("Negativo")**

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

Vamos a realizar un ejemplo para pedir dos números al usuario

Debemos indicar qué número es mayor

Debemos indicar si los números son iguales

Creamos un nuevo programa llamado **python05mayor.py**

**print("Numero mayor")**

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

**num1 = *int*(input())**

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

**num2 = *int*(input())**

**if (num1 > num2):**

**print ("El mayor es ", num1)**

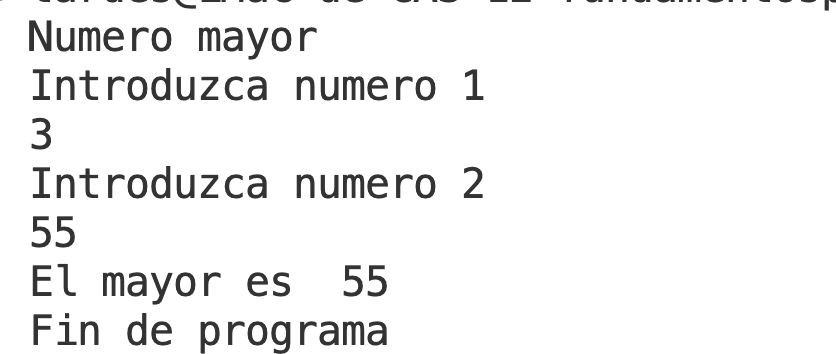
**elif (num1 == num2):**

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

**else:**

**print("El mayor es ", num2)**

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



CALCULAR LETRA DNI

* **Realizar una aplicación para conocer la letra del Documento Nacional de Identidad a través del número de DNI.**
* **La fórmula para calcular la letra del número del DNI se halla de la siguiente manera:**

**Se calcula el valor de la siguiente resta**

**( nº DNI - (ENTERO(nº DNI / 23) \* 23**

**Se mira la equivalencia en la siguiente tabla**

| 0=T | 4=G | 8=P | 12=N | 16=Q | 20=C |
| --- | --- | --- | --- | --- | --- |
| 1=R | 5=M | 9=D | 13=J | 17=V | 21=K |
| 2=W | 6=Y | 10=X | 14=Z | 18=H | 22=E |
| 3=A | 7=F | 11=B | 15=S | 19=L | 23=T |



**print("Calcular letra NIF")**

**print("Introduzca numero DNI")**

**dni = *int*(input())**

**#( nº DNI - (ENTERO(nº DNI / 23) \* 23**

**resultado = dni - (*int*(dni / 23) \* 23)**

**print(resultado)**

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

**print("T")**

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

**print("R")**

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

**print("W")**

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

**print("A")**

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

**print("G")**

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

**print("M")**

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

**print("Y")**

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

**print("F")**

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

**print("P")**

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

**print("D")**

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

**print("X")**

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

**print("B")**

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

**print("N")**

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

**print("J")**

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

**print("Z")**

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

**print("S")**

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

**print("Q")**

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

**print("V")**

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

**print("H")**

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

**print("L")**

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

**print("C")**

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

**print("K")**

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

**print("E")**

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

**print("T")**

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

**LIBRERIAS PYTHON**

Una librería es un conjunto de clases con sus funcionalidades.

Dichas librerías tienen un proposito y es ayudarnos en nuestros códigos.

Por ejemplo, tenemos librerías para acceder a ficheros.

Tenemos librerías para acceder a bases de datos

Debemos saber que existen dos tipos de librerías:

1) Librerías propias de Python integradas dentro de la instalación.

2) Librerías externas a la instalación de Python. (**pip**)

Para conocer este concepto, vamos a crear un simple programa que utilizará la librería **math**

Para utilizar librerías dentro de nuestro programa, tenemos dos métodos de realizarlo

**import libreria**

**libreria.metodo1()**

**libreria.metodo2()**

**print ("Librerias Python")**

**#LA UNICA FORMA DE HACER OPERACIONES MATEMATICAS COMPLEJAS**

**#ESTA EN UTILIZAR CLASES HERRAMIENTA. librerias**

**import math**

**resultado = 7 / 3**

**print(resultado)**

**print("Floor ", math.floor(resultado))**

**print("Ceil ", math.ceil(resultado))**

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

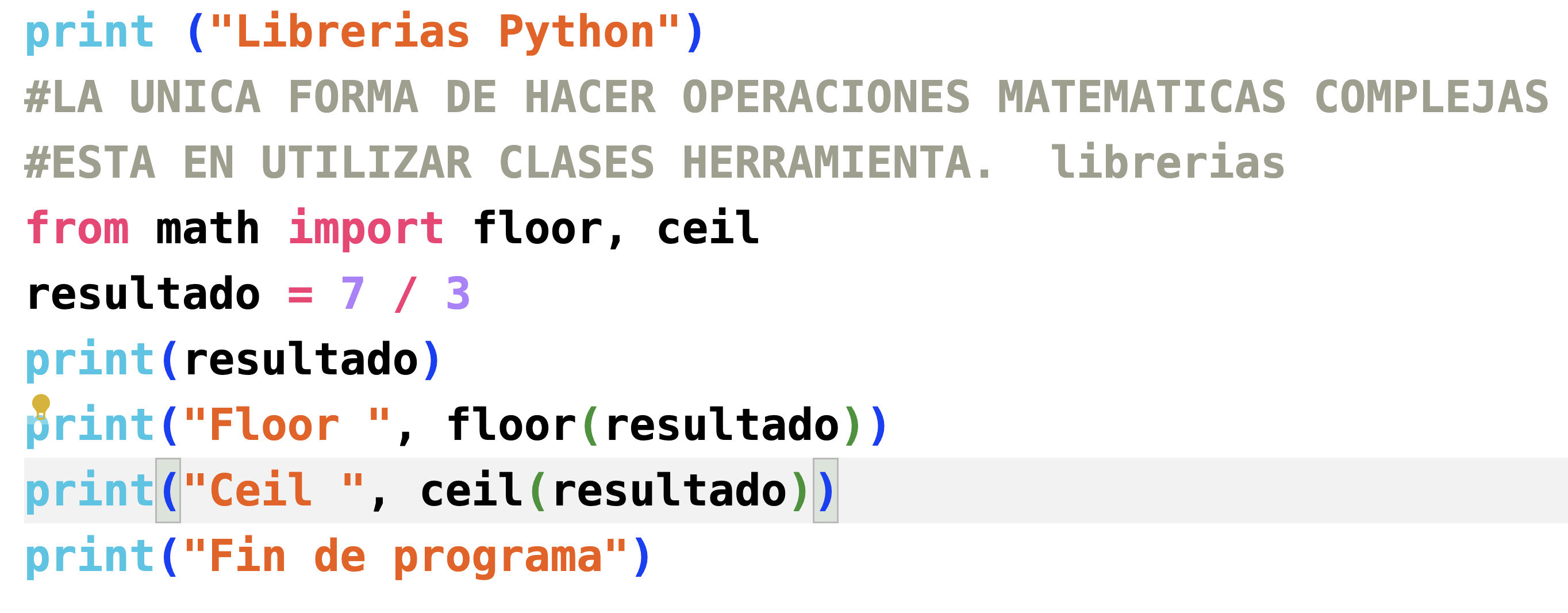
Tenemos otra sintaxis dónde nos traemos directamente los métodos de la librería

**from libreria import metodo1(), metodo2()**

**metodo1()**

**metodo2()**

Creamos un nuevo programa llamado **python07librerias.py**



**DEPURACION DE APLICACIONES**

La depuración de aplicaciones (Debug) nos permite «saber» qué está pasando en nuestro programa en tiempo de ejecución.

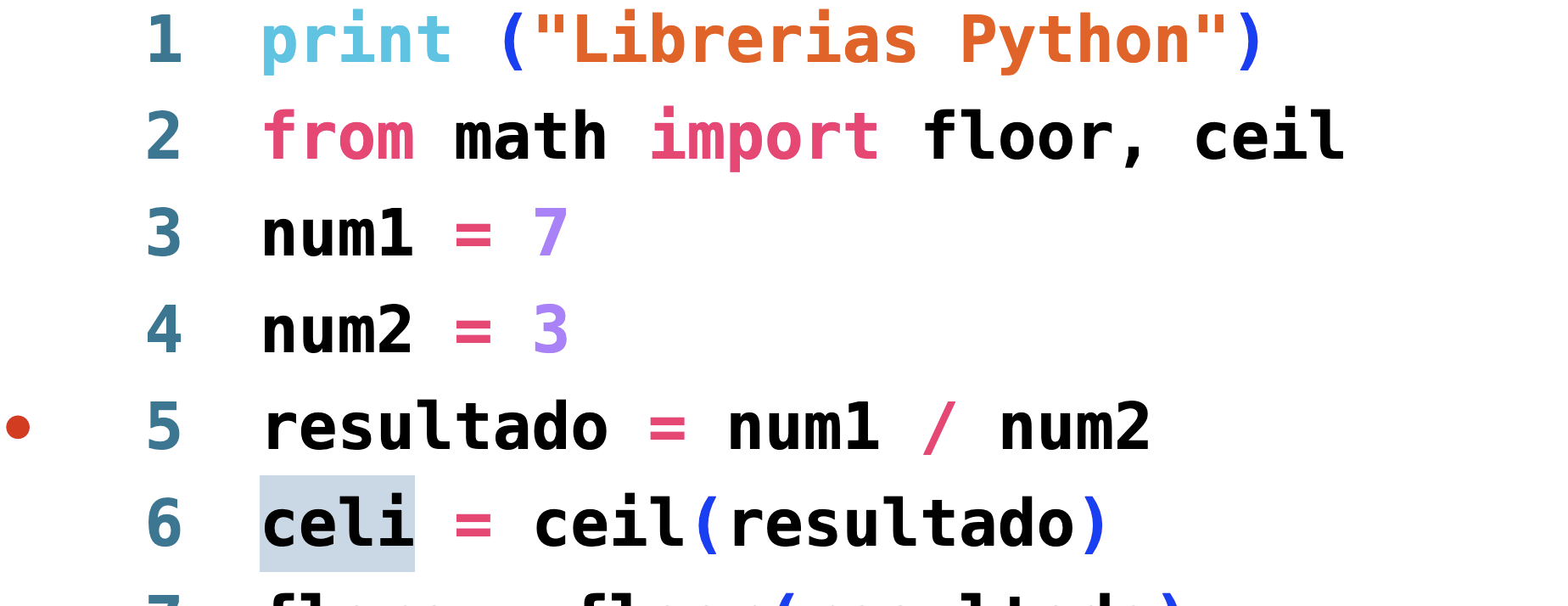
Super util cuando necesitamos saber el valor que están teniendo las variables en el código mientras ejecuta.

Sobre todo, para los errores lógicos.

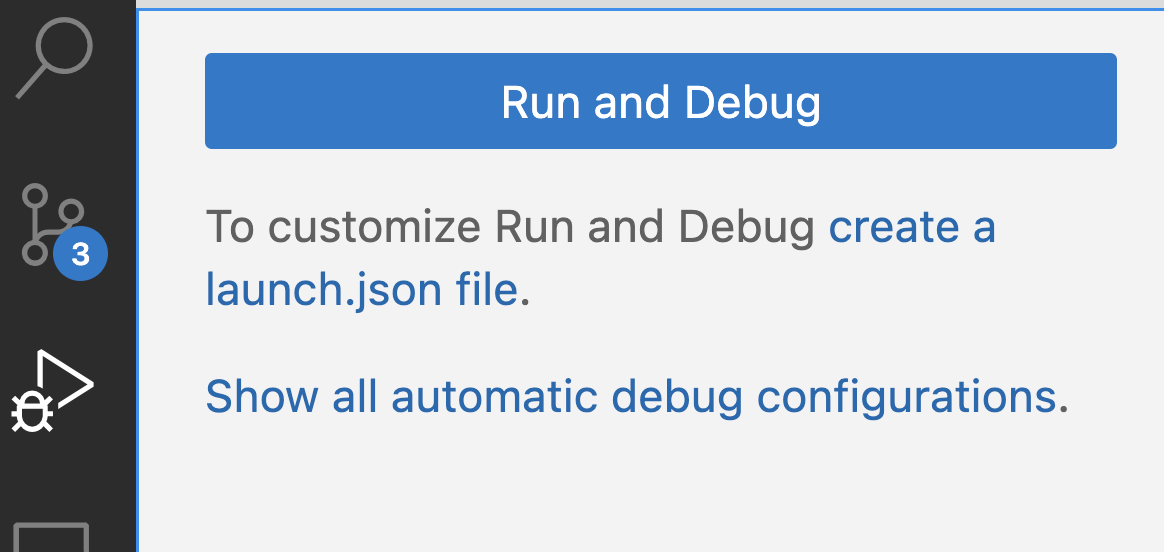
Vamos a visualizar cómo podemos averiguar qué está pasando en nuestro código cuando ejecutamos.

Para ello, debemos hacer lo siguiente:

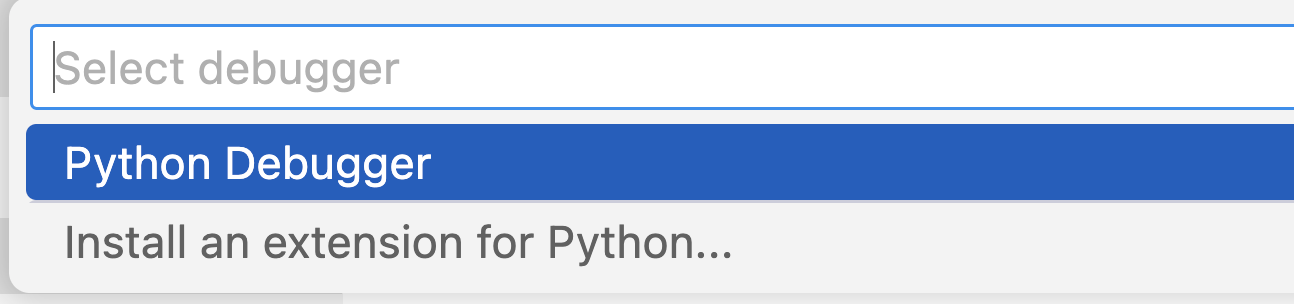
1) En la parte izquierda de las líneas de código, podemos poner un breakpoint, para que nuestra App se detenga en ese momento.

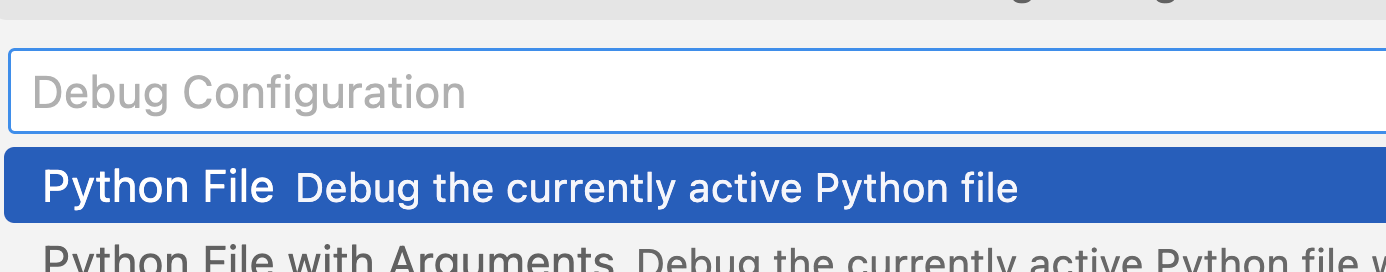


2) Debemos ejecutar con el bicho. Run and Debug.

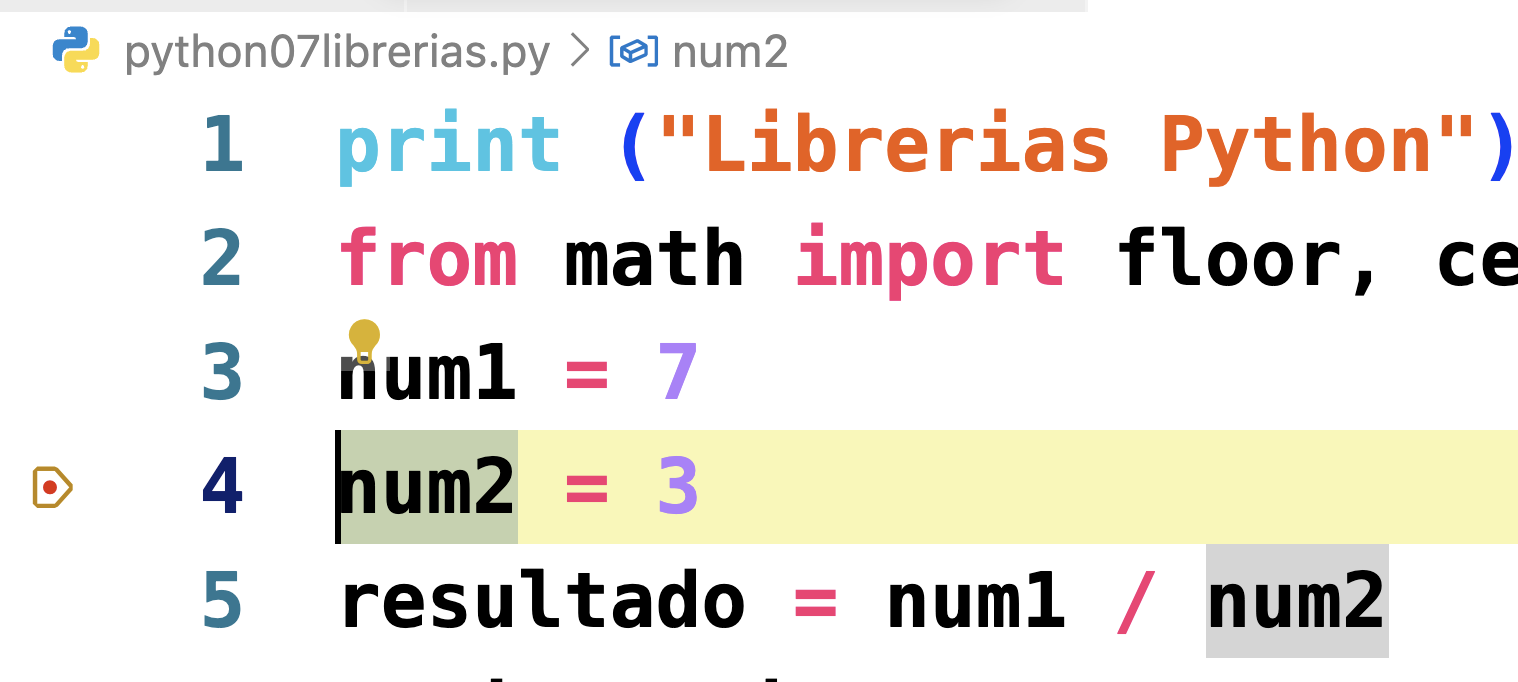


3) Indicamos el Python Debugger al pulsar en el botón





Una vez ejecutado, se queda detenido en la línea de mi breakpoint

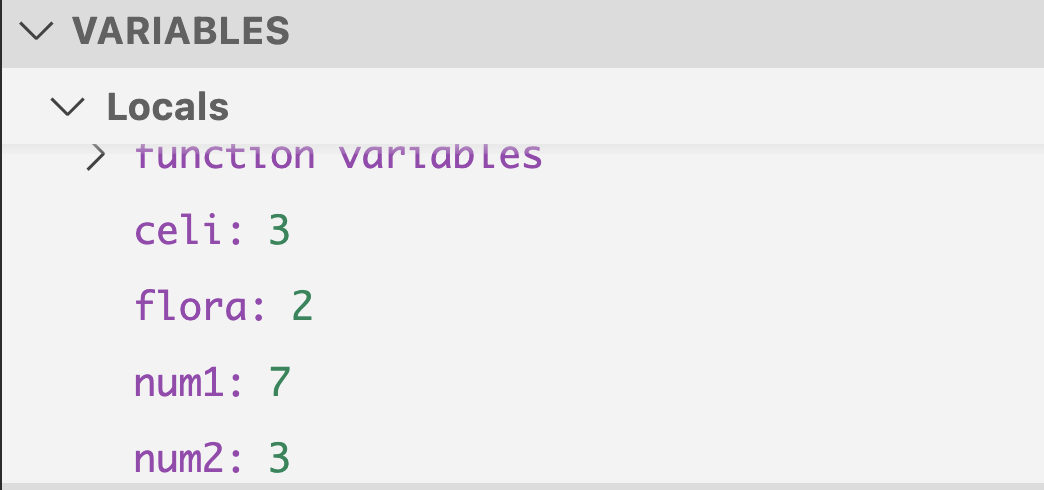


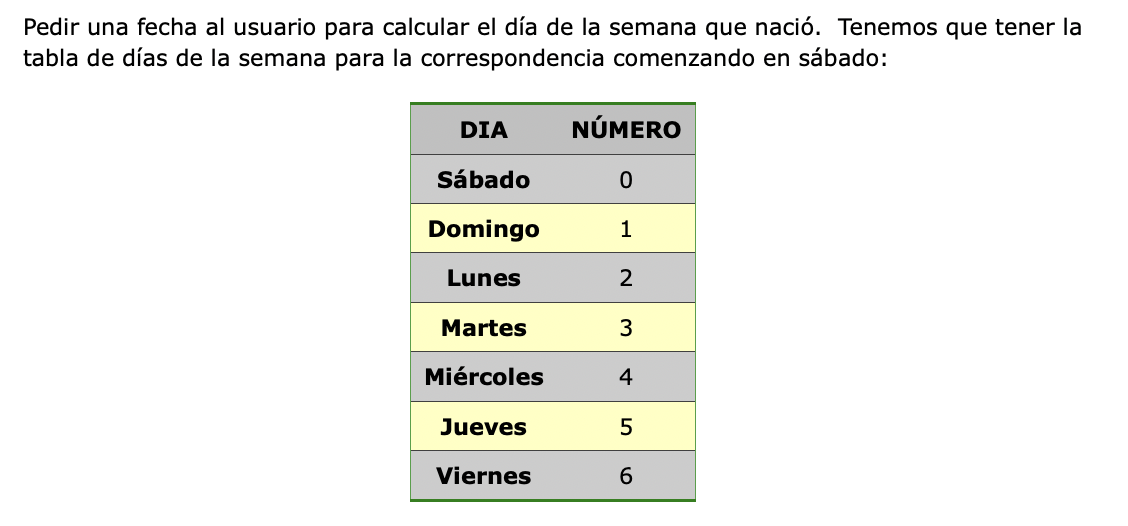
Yendo paso a paso, ya podemos averiguar dónde va nuestro programa y el valor de las variables.

Con el botón Step Over o F10, iremos avanzando en nuestro programa



Podremos ver en la zona de variables el valor que están recuperando en nuestro programa





Debemos pedir el **día**, el número de **mes** y el **año** que el usuario haya nacido.

A partir de esto datos hay que calcular lo siguiente para averiguar el día de la semana de nacimiento:

Ejemplo  15/06/1997

Hay que tener en cuenta el mes para realizar el cálculo, si el mes es Enero, el Mes será 13 y restaremos uno al año. Si el Mes es Febrero, el Mes será 14 y restaremos uno al año.

Para poder calcular las el número final de la semana debemos seguir los siguientes pasos:

1. Multiplicar el Mes más 1 por 3 y dividirlo entre 5

((6 + 1) \* 3) / 5  4

1. Dividir el año entre 4

1997 / 4  499

1. Dividir el año entre 100

1997 / 100  19

1. Dividir el año entre 400

1997 / 400  4

1. Sumar el dia, el doble del mes, el año, el resultado de la operación 1, el resultado de la operación 2, menos el resultado de la operación 3 más la operación 4 más 2.

15 + (6 \* 2) + 1997 + 4 + 499 - 19 + 4 + 2  2514

1. Dividir el resultado anterior entre 7.

2514 / 7  359

1. Restar el número del paso 5 con el número del paso 6 por 7.

2514 – (359 \* 7)  1

1. Miramos la tabla y vemos que el número 1 corresponde a **Domingo**

**print("Calcular dia semana")**

**print("Introduzca día de nacimiento")**

**dia = *int*(input())**

**print("Introduzca mes")**

**mes = *int*(input())**

**print("Su año de nacimiento")**

**anyo = *int*(input())**

**if (mes == 1):**

**mes = 13**

**anyo = anyo - 1**

**elif (mes == 2):**

**mes = 14**

**anyo = anyo - 1**

**#calculamos todo**

**ope1 = *int*(((mes+1) \* 3) / 5)**

**ope2 = *int*(anyo / 4)**

**ope3 = *int*(anyo / 100)**

**ope4 = *int*(anyo / 400)**

**# Sumar el dia, el doble del mes, el año,**

**# el resultado de la operación 1,**

**# el resultado de la operación 2,**

**# menos el resultado de la operación 3**

**# más la operación 4 más 2.**

**ope5 = dia + (mes \* 2) + anyo + ope1 + ope2 - ope3 + ope4 + 2**

**ope6 = *int*(ope5 / 7)**

**resultado = ope5 - (ope6 \* 7)**

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

**print("SABADO")**

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

**print("DOMINGO")**

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

**print("LUNES")**

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

**print("MARTES")**

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

**print("MIERCOLES")**

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

**print("JUEVES")**

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

**print("VIERNES")**

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

OPERADORES RELACIONALES

Estos operadores nos permiten realizar más de una pregunta a la vez y que dependa de las preguntas si se cumple la condición o no.

Tenemos dos palabras clave para utilizarlos:

**and**: Todas las condiciones de la pregunta deben cumplirse

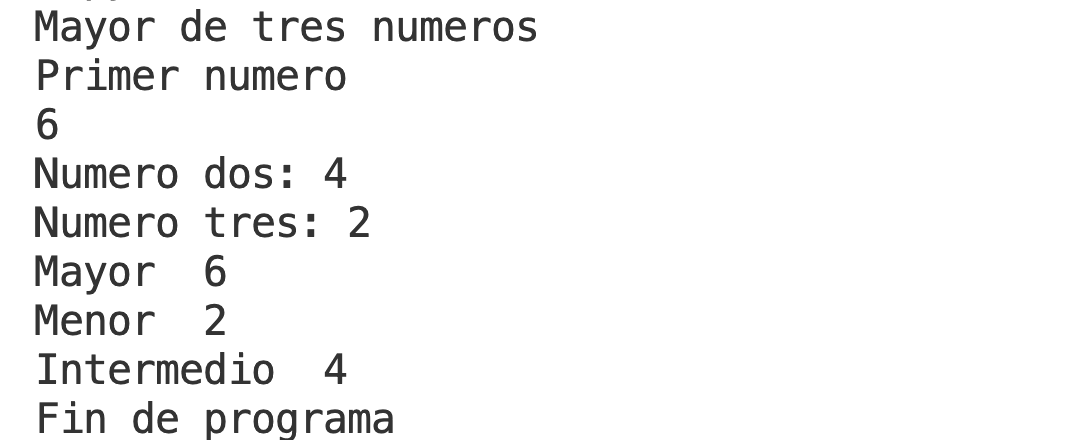
Lo puedo utilizar para rangos. Si un valor esta entre X y Z

**or:** Cualquier condición de la pregunta que sea true entra en el IF

OR lo puedo utilizar para preguntar sobre diferentes valores de una variable.

Vamos a realizar un ejemplo en el que pediremos tres números y mostraremos el mayor, el menor y el intermedio de los tres.

Creamos un nuevo programa llamado **python09mayortresnumeros.py**



**print("Mayor de tres numeros")**

**print("Primer numero")**

**num1 = *int*(input())**

**#directamente pedimos y guardamos**

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

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

**#organizar nuestro codigo**

**#las variables mejor no dejarlas en el programa por ahi tiradas**

**mayor = 0**

**menor = 0**

**intermedio = 0**

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

**mayor = num1**

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

**mayor = num2**

**else:**

**mayor = num3**

**#recuperamos el menor**

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

**menor = num1**

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

**menor = num2**

**else:**

**menor = num3**

**#6 , 4 , 2**

**# mayor: 6**

**# menor: 2**

**# 6 + 4 + 2 = 12**

**# 12 - 6 - 2 = 4**

**suma = num1 + num2 + num3**

**intermedio = suma - mayor - menor**

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

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

**print("Intermedio ", intermedio)**

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

**CALCULAR SALARIO**

* Realizar una aplicación que obtenga el salario neto de los empleados de una empresa teniendo en cuenta el número de horas trabajadas, el importe a pagar por hora y el lugar al que ha sido destinado el trabajador (establecido en Km).
* Cada hora extra se pagará 1.5€ más que las normales, (la hora extra comienza a computarse a partir de la 36 hora semanal trabajada)
* Sobre el salario base anterior, si el destino del empleado está:

Entre 101 Km y 900Km: La dieta será “NACIONAL”

* + - * 1. Por encima de 900Km: Dieta “INTERNACIONAL”
        2. Por debajo de 101 Km Dieta “PROVINCIAL”
* Sobre el precio final obtenido del salario:
  + 1. Cantidades menores o iguales a 250 € no soportan retención 0%
    2. Cantidades por encima de 250 € y menores o iguales a 500 € la retención es del 20%
    3. Cantidades por encima de 500 € la retención es del 50%
* Aplicaremos el IVA (16%) al salario bruto y se lo restaremos para conseguir el salario neto del trabajador.

**Al final debemos mostrar el siguiente informe:**

INTRODUZCA HORAS SEMANALES

51

INTRODUZCA IMPORTE HORA:

24

INTRODUZCA KILOMETROS:

269

---------------------------------------------------

HORAS TRABAJADAS: 51

HORAS EXTRAS: 15

IMPORTE DE LA HORA: 24

DISTANCIA EN KM: 269

DESTINO: NACIONAL

RETENCION: 50%

SALARIO BASE: 864.00

SALARIO HORAS EXTRA: 540.00

SALARIO BRUTO: 1404.00

IVA (16%): 224.64

---------------------------------------------------

SALARIO TOTAL: 1179.36

---------------------------------------------------

FIN DE PROGRAMA

**print("Calcular salario trabajador")**

**horas = *int*(input("Horas trabajadas: "))**

**importe\_hora = *int*(input("Importe hora: "))**

**km = *int*(input("Kilometros viajados: "))**

**horas\_extra = 0**

**salario\_base = 0**

**salario\_extra = 0**

**salario\_final = 0**

**dieta = ""**

**retencion = ""**

**iva = 0**

**if (horas > 36):**

**horas\_extra = horas - 36**

**salario\_base = importe\_hora \* 36**

**salario\_extra = (importe\_hora + 1.5) \* horas\_extra**

**else:**

**horas\_extra = 0**

**salario\_extra = 0**

**salario\_base = importe\_hora \* horas**

**if (km > 101 and km < 900):**

**dieta = "NACIONAL"**

**elif (km > 900):**

**dieta = "INTERNACIONAL"**

**else:**

**dieta = "PROVINCIAL"**

**salario\_final = salario\_base + salario\_extra**

**if (salario\_final <= 250):**

**retencion = "0%"**

**elif (salario\_final > 250 and salario\_final <= 500):**

**retencion = "20%"**

**else:**

**retencion = "50%"**

**iva = salario\_final \* 0.16**

**salario\_neto = salario\_final - iva**

**print("HORAS TRABAJADAS....", horas)**

**print("IMPORTE HORA.....", importe\_hora)**

**print("KILOMETROS.....", km)**

**print("Dietas....", dieta)**

**print("Retencion.....", retencion)**

**print("Horas extra....", horas\_extra)**

**print("SALARIO BASE.....", salario\_base)**

**print("Salario EXTRA....", salario\_extra)**

**print("Salario Bruto....", salario\_neto)**

**print("Iva: ", iva)**

**print("Salario final.....", salario\_final)**

**BUCLES**

Un bucle es una secuencia de código que se repite N veces mediante sus instrucciones.

Tenemos dos tipos de bucles:

1) **while:** Este tipo de bucle es condicional, lo que quiere decir que se repite mediante una condición y debe cumplirse la condición para finalizar.

Este tipo de bucles podría provocar un bucle infinito

Sintaxis:

while (condicion == True):

#Instrucciones

Codigo para False

2) **for:** Este tipo de bucle se repite un número determinado de veces. Le indicamos el inicio y el final y el bucle entra en ese inicio y finaliza en ese final.

Aquí estamos «protegidos» del bucle infinito.

La variable contador se declara dentro del propio bucle y solamente tendrá visibilidad dentro de dicho bucle.

Sintaxis:

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

#instrucciones

La variable contador siempre será CERO con esta sintaxis

Tenemos otra sintaxis en la que podemos iniciar el contador en un número inicial e indicar el número final

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

#instrucciones

Creamos un nuevo programa llamado **python11bucles.py**

**print("Ejemplos de bucles")**

**print("WHILE")**

**#Necesitamos una variable fuera del bucle**

**#para nuestra condicion**

**contador = 0**

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

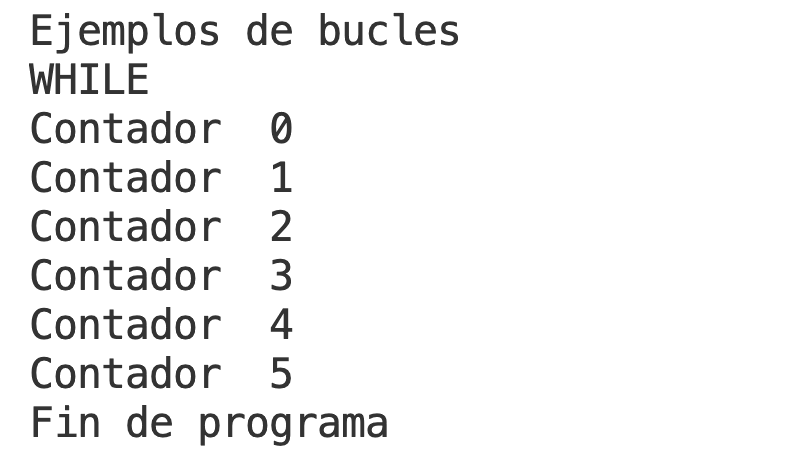
**print("Contador ", contador)**

**#Necesitamos cambiar el valor de "algo"**

**#para salir del bucle**

**contador = contador + 1**

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



**#En un bucle FOR, se declara la variable contador**

**#en la definicion del bucle**

**#Dicha variable suele llamarse i, z, k**

**#Hacemos un FOR de 0-5**

**print("FOR")**

**for i in range(5+1):**

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

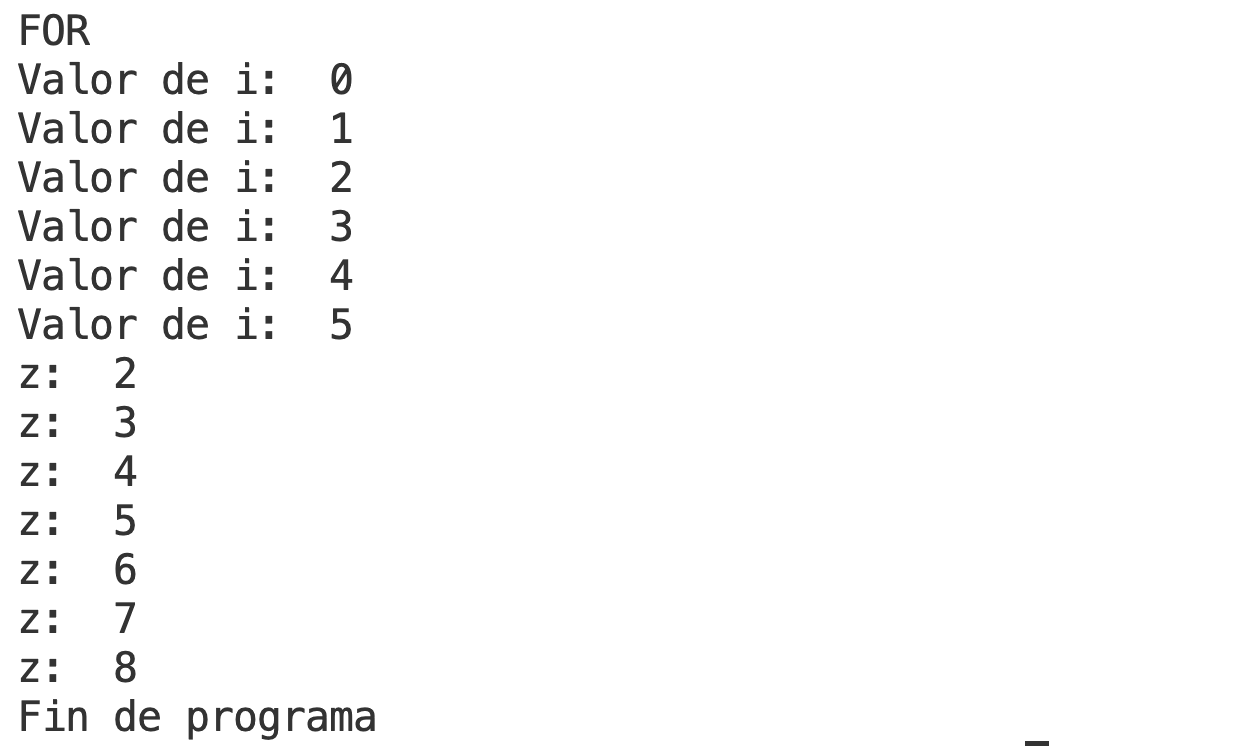
**#Tenemos la posibilidad de indicar un numero de inicio y**

**#un numero final. 2--8**

**for z in range(2, 8 + 1):**

**print("z: ", z)**

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

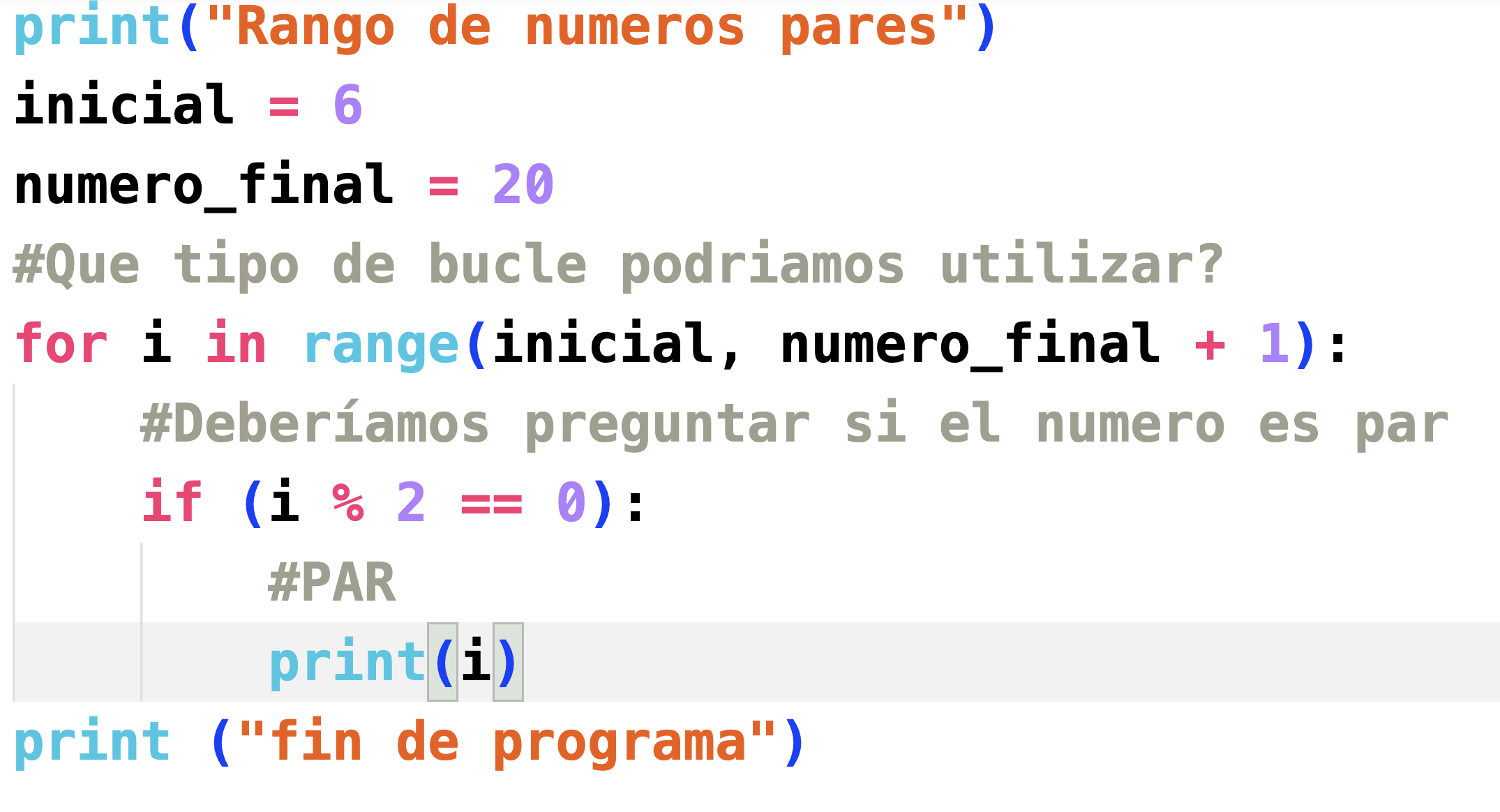


Vamos a realizar un programa en el que pediremos al usuario un número inicial y un número final.

Lo que queremos es mostrar los números pares que se encuentran entre dicho rango.

El operador para saber el resto de una división es %

Creamos un nuevo programa llamado **python12pares.py**



**CONJETURA DE COLLATZ**

Todo número entero positivo siempre llegará a ser 1 siguiendo dos condiciones:

\* Si el número es par, dividimos entre 2

\* Si el número es impar, multiplicamos por 3 y sumamos 1

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

Creando otro programa llamado **python13collatz.py**

**print("Conjetura Collatz")**

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

**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")**

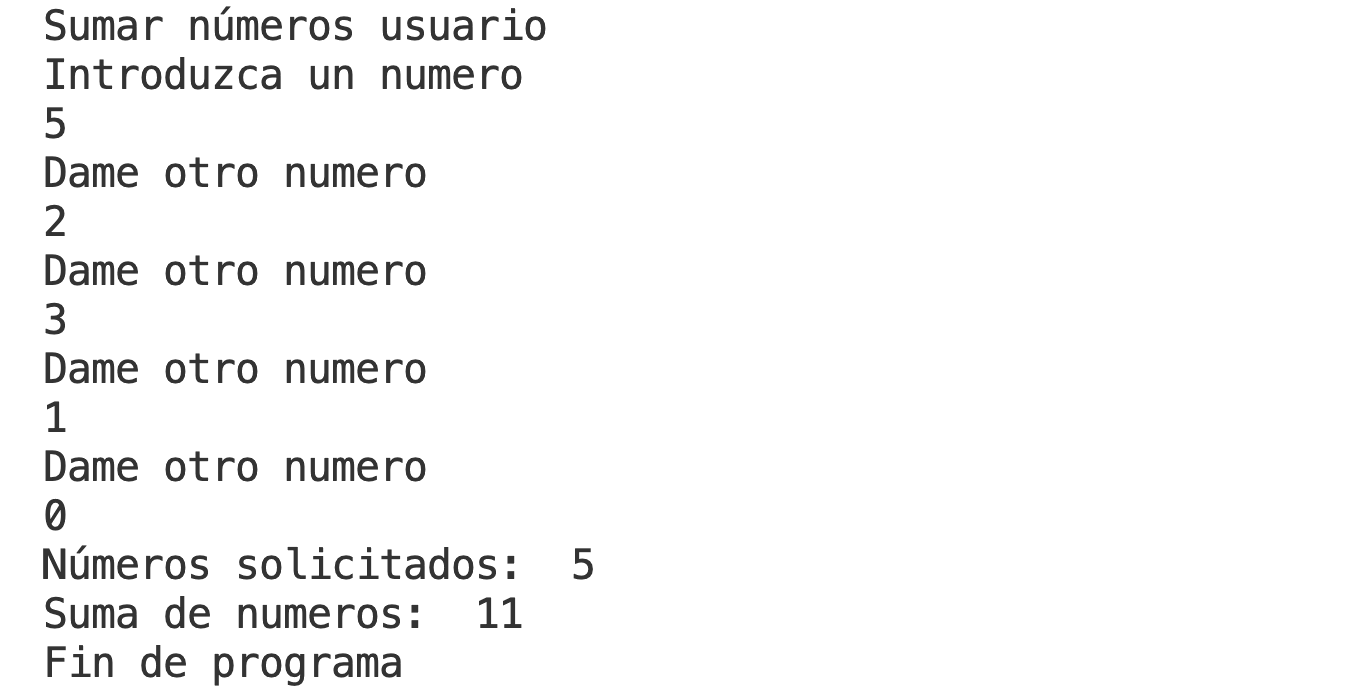
Necesito un programa que pedirá números al usuario todo el tiempo.

El programa finalizará cuando el usuario escriba un Cero.

El resultado del programa será el siguiente:

Números introducidos: 6

La suma de todos los números es: 80



Creamos un nuevo programa llamado

**python14pedirnumeros.py**

**print("Sumar números usuario")**

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

**numero = *int*(input()) #5**

**contador = 1**

**suma = numero**

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

**print("Dame otro numero")**

**numero = *int*(input()) #2**

**suma = suma + numero**

**contador = contador + 1**

**print("Números solicitados: ", contador)**

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

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

Necesito un programa que nos muestre la tabla de multiplicar de un número que pediremos al usuario.

5\*1 = 5

5\*2 = 10

5\*3 = 15

…

5\*10 = 50

Creamos un nuevo programa llamado

**python15tablamultiplicar.py**

**print("Tabla multiplicar")**

**print("Introduce un numero")**

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

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

**resultado = numero \* i**

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

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

**CLASE STRING**

Una clase es la definición de un objeto.

Una clase está compuesta por propiedades y métodos.

Dependiendo del tipo de clase, tendremos unos métodos u otros métodos que serán herramientas para manejar dicho objeto.

La clase string es la clase que permite almacenar y manejar textos.

Por ejemplo, podemos convertir un texto a mayúsculas.

Un string no deja de ser un conjunto de letras.

Podemos averiguar el tamaño de un conjunto mediante la función **len()**

String diferencia mayúsculas de minúsculas

texto = “Lenguaje Python”

longitudTexto = 15 → **len(texto)**

Debemos saber que los string siempre comienzan en posición CERO.

Cada letra, estará en una posición, la primera letra será la posición 0 y la última letra será len() - 1

texto = “Lenguaje Python”

longitudTexto = 15 → **len(texto)**

Tenemos la posibilidad de recuperar la posición de cada letra por su índice

texto[0] = L

texto[1] = e

…

texto[14] = n

Tenemos una serie de métodos para trabajar con la clase string.

\* **upper():** Convierte un texto a mayúsculas

\* **lower()**: Convierte un texto a minúsculas

\* **find(«contenido»)**: Busca un texto dentro del string y nos devuelve su posición. Si no lo encuentra, nos devuelve -1

\* **rfind()**: Busca un texto dentro del string y nos devuelve su posición, empezando por la derecha. Si no lo encuentra, nos devuelve -1

Dentro de Python, tenemos POO, lo que quiere decir que se aplican ciertas técnicas de dicha programación

Una de las técnicas es tener Polimorfismo, varias formas para un método.

\* **find(«texto», indice)**: Busca un texto dentro del string a partir del índice y nos devuelve su posición. Si no lo encuentra, nos devuelve -1

\* **startswith():** Devuelve True si el string comienza con un texto determinado

\* **endswith():** Devuelve True si el string finaliza con un texto determinado

\* **replace(«antiguo», «nuevo»):** Reemplaza los caracteres antiguos con los nuevos en el string

\* **count(«texto»):** Cuenta el numero de veces que se repite un texto en un string

Tenemos métodos para saber el tipo de contenido del string

\* isdigit(**):** Indica si es un número el texto

\* isalpha(**):** Indicasi el texto son letras

\* isnum(**):** Indica si el texto contiene caracteres numéricos

\* isupper**():** Si está en mayúsculas

\* islower(): Si el texto está en minúsculas

Dentro de Python también tenemos la posibilidad de extraer subcadenas de un texto. **PYTHON SLICING**

texto = «primer python»

Tenemos dos posibilidades:

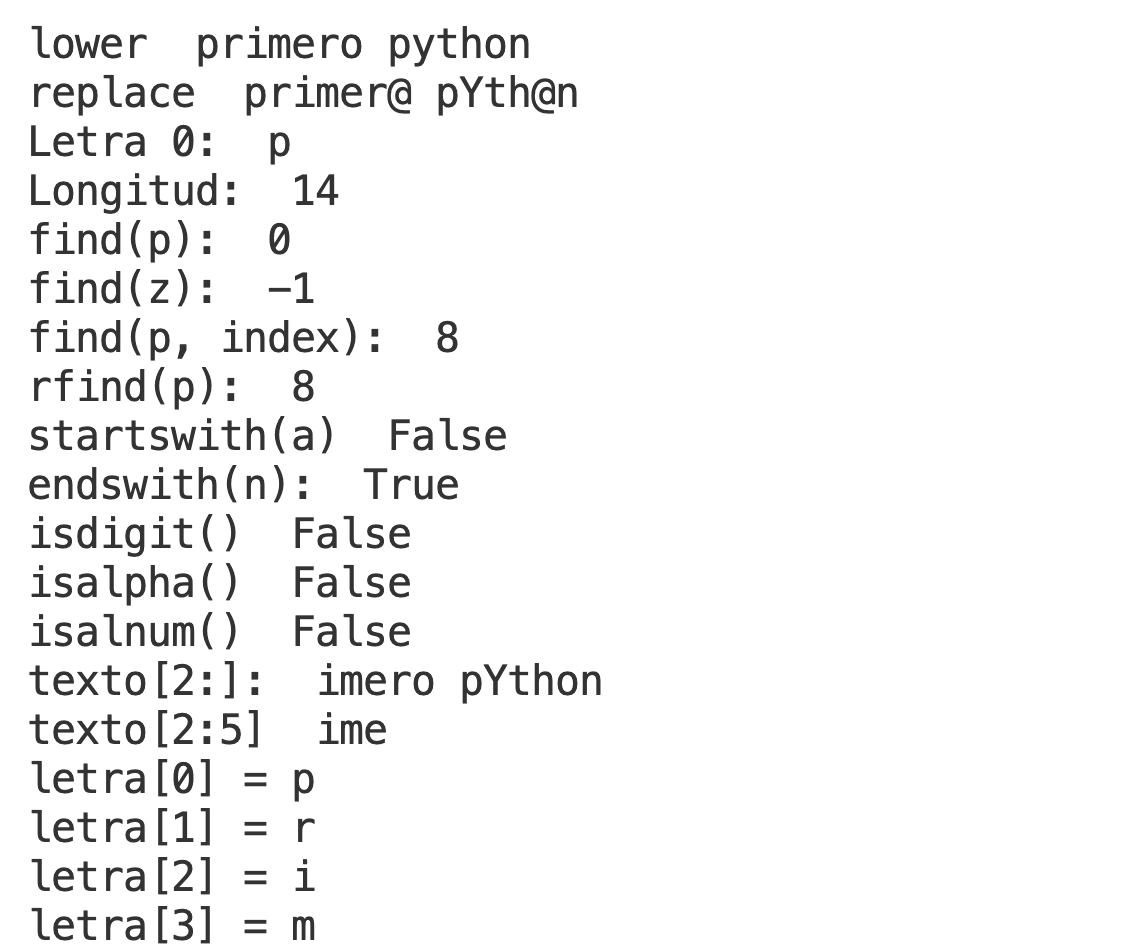
1) Queremos la subcadena desde una posicion concreta

texto[8:] – – > **ython**

2) queremos una subcadena desde una posición inicial hasta una posición final: objeto[inicio:fin]

texto[0:2] – – > **pr**

Creamos un nuevo programa llamado **python16string.py**



**print("Ejemplos clase STRING")**

**texto = "primero pYthon"**

**#VAMOS A IR PROBANDO DIFENTES METODOS**

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

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

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

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

**print("Longitud: ", len(texto))**

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

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

**# texto.find("p",1)**

**# texto.find("p",upper)**

**# texto.find("p",lower)**

**#SOBRECARGA DE UN METODO**

**print("find(p, index): ", 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()) #letras y numeros**

**#Probamos SLICING, llamado Substring**

**#Recuperar una parte de un texto**

**subcadena = texto[2: ]**

**print("texto[2:]: ", subcadena)**

**#En python tambien podemos recuperar desde una posicion**

**#hasta otra posicion**

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

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

**#Podemos recorrer cada caracter con un bucle**

**longitud = len(texto)**

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

**letra = texto[i]**

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

**#Incluso podemos validar lo que nos ofrece un usuario...**

**print("Introduce un numero")**

**dato = input() #"22"**

**if (dato.isdigit() == True):**

**print("Me has dado un numero")**

**else:**

**print("Que me des un numero!!!")**

No quiero bucles. Solamente preguntas IF

Necesito un ejemplo en el que escribiremos un email. Se lo pedimos al usuario.

**python17email.py**

Debemos indicar si tiene algún error:

\* Que el email contenga @

\* Que el email contenga un punto

\* @ ni al principio ni al final

\* Que el email solamente contenga una @

\* Que exista un punto después de la @

\* Que el dominio sea de 2 a 3 caracteres (es/com/org)

**print("Validación de Email")**

**print("Introduzca su email")**

**email = input()**

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

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

**print("No existe @ en el email")**

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

**print("El email no tiene .")**

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

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

**elif (email.find("@") != email.rfind("@")):**

**print("Existe mas de una @")**

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

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

**else:**

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

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

**longDominio = len(dominio)**

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

**print("Email correcto")**

**else:**

**print("El dominio debe ser de 2 a 4 caracteres")**

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

Necesito un programa que pedirá una cifra de números

Dame números: 1234

Nosotros, debemos realizar dos características en el programa:

1) Comprobar que nos han dado números. Si no son números, mostramos un mensaje de error.

2) Si son números, mostramos la suma de cada número del texto: 1 + 2 + 3 + 4 = 10



**print("Sumar números letras")**

**print("Introduce una cifra")**

**textoNumeros = input()**

**#AVERIGUAMOS SI SON NUMEROS O NO**

**if (textoNumeros.isdigit() == False):**

**print("Esto no son numeros: ", textoNumeros)**

**else:**

**#1+2+3+4**

**suma = 0**

**#Recorremos cada caracter del texto**

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

**#Necesito cada letra del texto**

**letra = textoNumeros[i]**

**#Convertimos la letra a numero**

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

**suma = suma + numero**

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

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

**VALIDACIÓN ISBN**

1. Realizar una aplicación para comprobar si el ISBN introducido por teclado es válido.
2. Debemos comprobar que el número introducido tiene 10 caracteres.

EJEMPLO DE NUMERO ISBN CORRECTO:

| **8441513929** |
| --- |

1.- Se descompone la cadena y se multiplica cada número por la posición que ocupa en la cadena:

8 \* 1

4 \* 2

4 \* 3

1 \* 4

5 \* 5

.

.

.

9 \* 10

2.- La suma de todas estas multiplicaciones se divide entre 11, y si el resto es cero, el número ISBN es correcto.

**print("Validar ISBN")**

**print("Introduzca ISBN")**

**isbn = input()**

**if (len(isbn) != 10):**

**print("El ISBN debe tener 10 caracteres")**

**else:**

**suma = 0**

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

**#recuperamos cada caracter**

**caracter = isbn[i]**

**#convertimos a numero el caracter**

**numero = *int*(caracter)**

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

**suma = suma + multi**

**#Preguntamos si la suma es divisible entre 11**

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

**print("Numero ISBN correcto")**

**else:**

**print("El número introducido no es ISBN")**

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

**FUNCIONES/METODOS DENTRO DE PYTHON**

Una función es un método.

Un método es una función.

Son herramientas que contienen los programas de Python y sus clases también

Un método de una clase Python: (string) (**upper()**)

Funciones de Python: **len(texto)**

Cualquier programa se ejecuta y UTILIZA funciones/métodos de Python.

Los métodos o funciones hacen que nuestro código NO sea lineal, además podemos reutilizar las funciones.

Actualmente, hemos validado un email, hemos validado un DNI. Todo estupendo.

Si ahora mismo, yo quiero validar un EMAIL, qué tendría que hacer?

El código lo separaremos en diferentes Programas.

Tendremos programas con métodos y programas con ejecución/llamadas de dichos métodos

También podemos tener en un mismo programa, métodos y también la parte principal de la ejecución.

Nuestro programa Python (MAIN)

print(«Validacion de email»)

print(«Dame un Email»)

email = input()

#Llamada a una función

**validarEmail(email)**

En otro lugar de nuestro CODIGO/FICHERO

validarEmail(emailRecibido):

#COMPROBAMOS LAS COSAS

En Python, para declarar métodos se utiliza la palabra clave **def** y a continuación el nombre del método

Los métodos pueden estar en el mismo fichero o en otro ficheros distinto.

Si los métodos están en el mismo fichero, lo primero que se escriben son dichos métodos.

Los métodos/funciones pueden recibir parámetros

Vamos a visualizar la sintaxis mediante un programa.

Creamos un nuevo programa llamado

**python20metodos.py**

**#EN UN PROGRAMA QUE DESEAMOS EJECUTAR**

**#LOS METODOS SE ESCRIBEN AL INICIO**

**#LA SINTAXIS DE LOS METODOS SE ESCRIBE**

**#CON LA SEGUNDA INICIAL Y SIGUIENTES EN MAYUSCULA**

**# primer()**

**# primerMetodo()**

**# primerMetodoMartes()**

***def* primerMetodo():**

**#ESTE CODIGO JAMAS SE EJECUTA**

**#SI NO LO LLAMAMOS EXPLICITAMENTE**

**print("Primer metodo")**

***def* segundoMetodo():**

**print("Segundo metodo")**

**#ESTO ES MI PROGRAMA PRINCIPAL**

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

**#En el programa principal, llamamos a los metodos**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

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

**PARAMETROS EN LOS METODOS**

Un método puede recibir parámetros, es decir, información que necesita el método para poder ser ejecutado.

Los parámetros se reciben dentro de la declaración del método entre los paréntesis.

Si recibimos más de un parámetro, irán separados con comas cada uno.

Las variables que enviamos y las variables que declaramos NO tienen porque coincidir, son distintos nombres y no tienen nada que ver entre sí.

Los parámetros solamente se pueden UTILIZAR dentro del método.

Ejemplo:

def metodoSaludar(**nombre**):

print(«Bienvenido» + **nombre**)

---------------------------------------------------------------

print(«Hola que tal, estoy en un main»)

name = «Paco»

metodoSaludar( name )

Metodos con parámetros

**#EN UN PROGRAMA QUE DESEAMOS EJECUTAR**

**#LOS METODOS SE ESCRIBEN AL INICIO**

**#LA SINTAXIS DE LOS METODOS SE ESCRIBE**

**#CON LA SEGUNDA INICIAL Y SIGUIENTES EN MAYUSCULA**

**# primer()**

**# primerMetodo()**

**# primerMetodoMartes()**

***def* primerMetodo():**

**#ESTE CODIGO JAMAS SE EJECUTA**

**#SI NO LO LLAMAMOS EXPLICITAMENTE**

**print("Primer metodo")**

***def* segundoMetodo():**

**print("Segundo metodo")**

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

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

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

**print("Ha sido un placer hoy " + dia + ", Mr/Mrs " + nombre)**

**#ESTO ES MI PROGRAMA PRINCIPAL**

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

**#En el programa principal, llamamos a los metodos**

**saludar("Alumno")**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

**despedirse("Alumno", "Martes")**

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

Los métodos los podemos dividir en dos grupos:

\* **Métodos de acción**: Ejecutan una serie de acciones

\* Métodos **return**: Ejecutan una serie de acciones y devuelven un valor después de ejecutar dichas acciones.

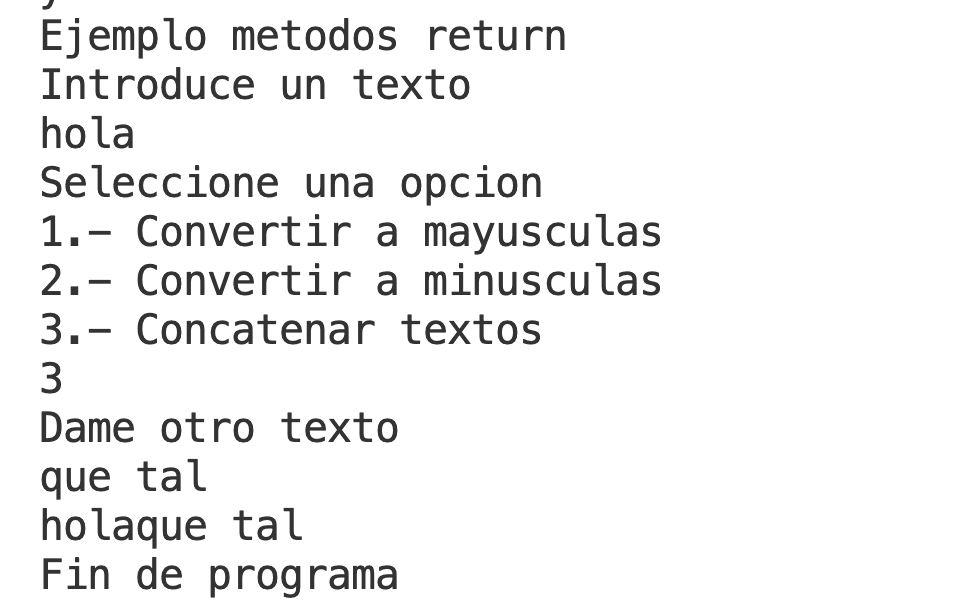
Ejemplo:

Acción: **print()**

Return: **len()**

Creamos un nuevo programa llamado

**python21metodosreturn.py**



**#Este metodo convierte a mayusculas**

**#y nos devuelve el texto a mayusculas**

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

**return texto.upper()**

**#Otro metodo que convierte a minusculas**

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

**return texto.lower()**

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

**resultado = texto1 + texto2**

**return resultado**

***def* mostrarMenu():**

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

**print("1.- Convertir a mayusculas")**

**print("2.- Convertir a minusculas")**

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

**#PROGRAMA PRINCIPAL MAIN**

**print("Ejemplo metodos return")**

**print("Introduce un texto")**

**valor = input()**

**mostrarMenu()**

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

**resultado = ""**

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

**resultado = convertirMayusculas(valor)**

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

**resultado = convertirMinusculas(valor)**

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

**print("Dame otro texto")**

**otro = input()**

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

**print(resultado)**

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

Creamos un nuevo programa llamado

**python22calculadora.py**

En el programa main

1) Pediremos dos números al usuario

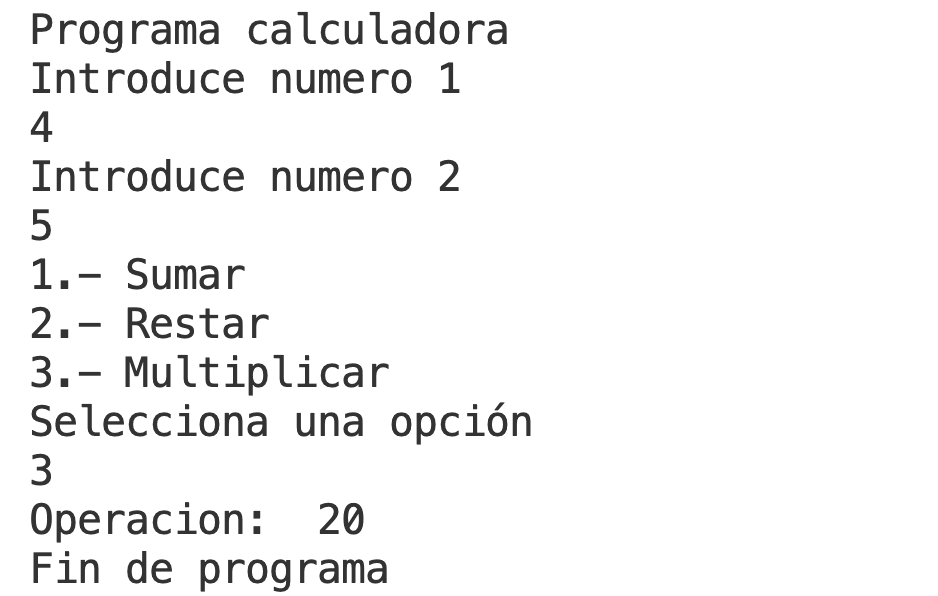
2) Mostraremos un menú con las siguientes opciones

1.- Sumar

2.- Restar

3.- Multiplicar

Tendremos cuatro funciones, **tres return** con las operaciones, y otra para mostrar el menú



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

**suma = num1 + num2**

**return suma**

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

**return num1 - num2**

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

**return num1 \* num2**

***def* menuCalculadora():**

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

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

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

**print("Selecciona una opción")**

**#Programa principal MAIN**

**print("Programa calculadora")**

**print("Introduce numero 1")**

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

**print("Introduce numero 2")**

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

**menuCalculadora()**

**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("Opción no valida")**

**print("Operacion: ", operacion)**

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

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

**VERSION 2**

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

**suma = num1 + num2**

**return suma**

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

**return num1 - num2**

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

**return num1 \* num2**

***def* menuCalculadora():**

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

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

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

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

**print("Selecciona una opción")**

**#Programa principal MAIN**

**print("Programa calculadora")**

**print("Introduce numero 1")**

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

**print("Introduce numero 2")**

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

**#Quiero mostrar el menu calculadora, hasta cuando???**

**#Cuando el usuario ponga SALIR ( 0 ), salimos del bucle**

**#Vamos a dar un valor por defecto a opcion para que**

**#entre en el bucle**

**opcion = -1**

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

**menuCalculadora()**

**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("Opción no valida")**

**print("Operacion: ", operacion)**

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

**Versión 3:**

\* Si el usuario no introduce números, pedimos números hasta que sea correcto

\* Creamos otra opción más para que el usuario pueda introducir nuevos números.

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

**suma = num1 + num2**

**return suma**

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

**return num1 - num2**

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

**return num1 \* num2**

***def* menuCalculadora():**

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

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

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

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

**print("4.- Introducir nuevos números")**

**print("Selecciona una opción")**

**#Creamos un metodo que devolverá el número introducido**

**#En este método, tendremos un bucle infinito hasta que**

**#nos de un número**

***def* getNumero():**

**print("Introduzca un número")**

**#ALMACENAMOS LO QUE ESCRIBA EL USUARIO**

**#EN UNA VARIABLE string**

**aux = input()**

**#Entramos en un bucle mientras que NO sean NUMEROS**

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

**print("Eso no era un numero")**

**print("Introduzca número")**

**aux = input()**

**#Convertimos el texto a numero**

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

**#Devolvemos el numero desde el metodo**

**return num**

**#Programa principal MAIN**

**print("Programa calculadora")**

**numero1 = getNumero()**

**numero2 = getNumero()**

**#Quiero mostrar el menu calculadora, hasta cuando???**

**#Cuando el usuario ponga SALIR ( 0 ), salimos del bucle**

**#Vamos a dar un valor por defecto a opcion para que**

**#entre en el bucle**

**opcion = -1**

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

**menuCalculadora()**

**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):**

**numero1 = getNumero()**

**numero2 = getNumero()**

**else:**

**print("Opción no valida")**

**print("Operacion: ", operacion)**

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

**LIBRERIAS EN PYTHON**

Python ofrece una serie de librerías que podemos utilizar, por ejemplo, la librería **math**

La librería **math** contiene una serie de métodos para trabajar con ella.

import math

from math import trunc, floor

Acabamos de crear un ejemplo con un programa y métodos, todo junto en el mismo fichero.

Ahora mismo, cada vez que yo quiera utilizar **getNumero**, necesitamos copiar el método **def getNumero()** a otro programa.

El concepto de librería está en separar los métodos de los programas.

Lo que haremos a partir de ahora es tener un fichero que será nuestro **main/program** que será el que ejecutemos.

Tendremos otros ficheros que serán las librerías con los métodos que decidamos.

**LIBRERIAS**

def metodo1():

CODIGO

def metodo2():

CODIGO

**MAIN**

from **LIBRERIAS** import metodo1(), metodo2()

print («Bienvenido a mi programa»)

dato = input()

metodo1()

**Los nombre de librerías serán los nombre de los ficheros de python**

En nuestro programa trabajaremos así a partir de ahora:

\* libreriaXX: La clase con los metodos def

\* pythonXX: Esto será nuestro programa Main/Program de ejecución

Lo primero será llevarnos nuestro ejemplo de Calculadora, a este nuevo panorama.

Creamos un nuevo fichero llamado

**libreria23matematicas.py**

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

**suma = num1 + num2**

**return suma**

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

**return num1 - num2**

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

**return num1 \* num2**

***def* menuCalculadora():**

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

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

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

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

**print("4.- Introducir nuevos números")**

**print("Selecciona una opción")**

**#Creamos un metodo que devolverá el número introducido**

**#En este método, tendremos un bucle infinito hasta que**

**#nos de un número**

***def* getNumero():**

**print("Introduzca un número")**

**#ALMACENAMOS LO QUE ESCRIBA EL USUARIO**

**#EN UNA VARIABLE string**

**aux = input()**

**#Entramos en un bucle mientras que NO sean NUMEROS**

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

**print("Eso no era un numero")**

**print("Introduzca número")**

**aux = input()**

**#Convertimos el texto a numero**

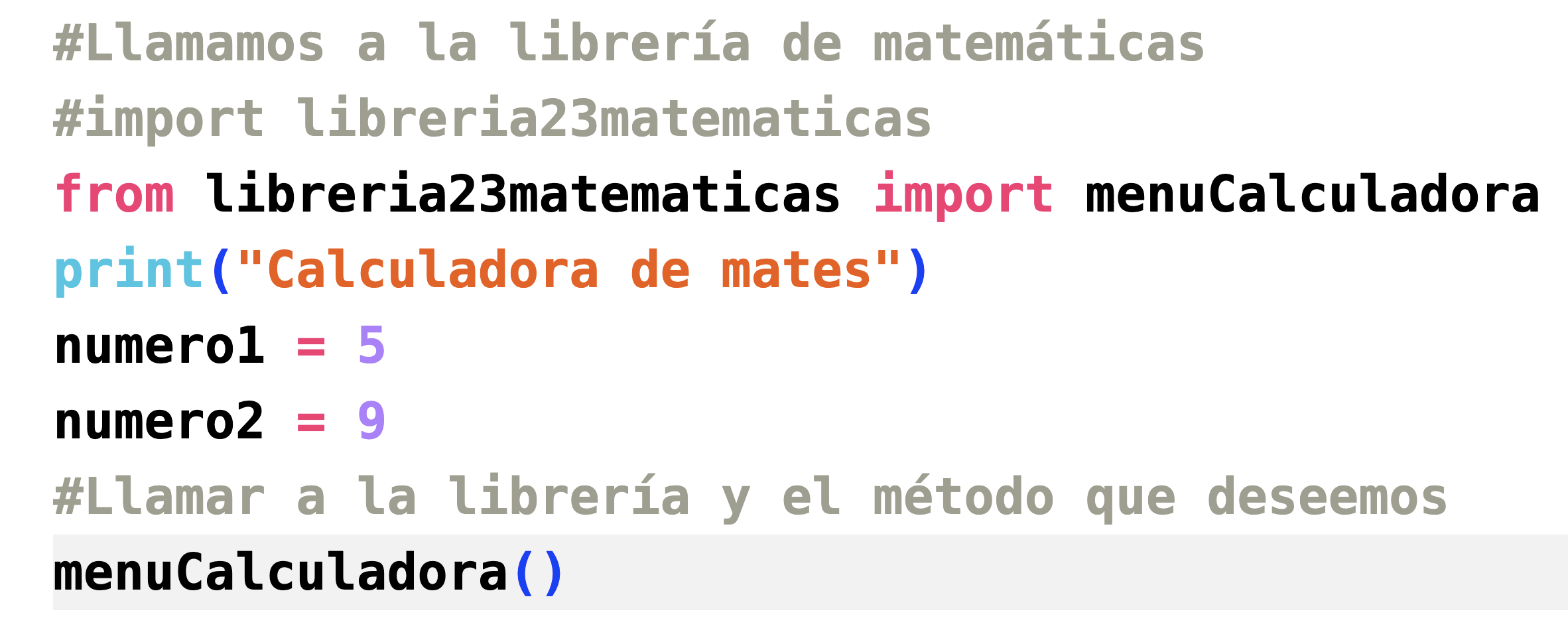
**num = *int*(aux)**

**#Devolvemos el numero desde el metodo**

**return num**

Creamos un nuevo programa llamado

**python23matematicas.py**



El siguiente ejemplo que necesitamos es ir copiando código y probar este nuevo escenario.

Vamos a tener una librería que se encargará de las

Validaciones.

1) Tendremos un método que será validar el ISBN. True/False 8441513929

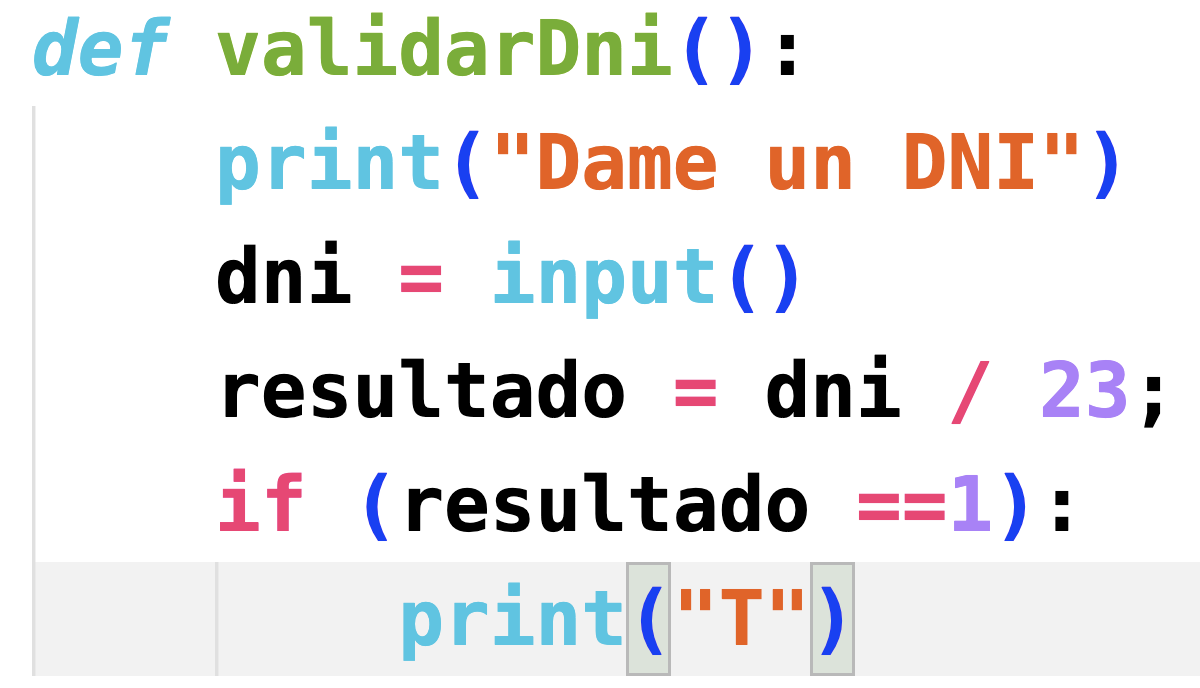
2) Un método que recibirá el número del DNI y nos devolverá la LETRA.

3) Tendremos un método que nos dirá si un DNI es correcto. Enviaremos TODO el DNI: 12345678Z y nos díra si es True/False

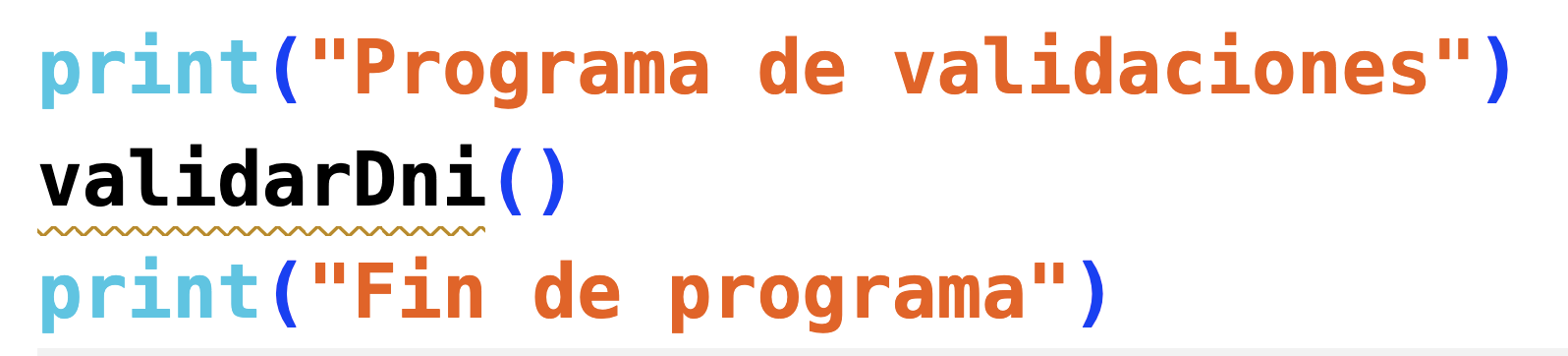
**libreria24validaciones, python24validaciones**

Ahora, me gustaría separar la capa visual de la capa lógica.

CAPA VISUAL Y LOGICA



PROGRAMA MAIN



En realidad, deberíamos aislar lo que queremos del dibujo en el que estamos.



**LIBRERIA24VALIDACIONES.PY**

***def* validarDni(*dni*): #12345678Z**

**#separamos el numero de la letra que nos han dado**

**numeros = dni[0: len(dni) - 1]**

**letraEscrita = dni[len(dni) -1 :]**

**numeros = *int*(numeros)**

**letraCorrecta = getLetraDni(numeros)**

**if (letraCorrecta.upper() == letraEscrita.upper()):**

**return True**

**else:**

**return False**

***def* getLetraDni(*numeroDni*):**

**resultado = numeroDni - (*int*(numeroDni / 23) \* 23)**

**muestraLetras = "TRWAGMYFPDXBNJZSQVHLCKET";**

**letra = muestraLetras[resultado]**

**return letra**

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

**if (len(isbn) != 10):**

**return False**

**else:**

**suma = 0**

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

**#recuperamos cada caracter**

**caracter = isbn[i]**

**#convertimos a numero el caracter**

**numero = *int*(caracter)**

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

**suma = suma + multi**

**#Preguntamos si la suma es divisible entre 11**

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

**return True**

**else:**

**return False**

**PYTHON24VALIDACIONES.PY**

**import libreria24validaciones**

**print("Main de validaciones")**

**print("Introduzca un Dni")**

**dni = input()**

**respuesta = libreria24validaciones.validarDni(dni)**

**if (respuesta == True):**

**print("El dni es correcto")**

**else:**

**print("El dni es INCORRECTO")**

**#numDni = int(input())**

**#letra = libreria24validaciones.getLetraDni(numDni)**

**#print(letra) #Z**

**#separamos el numero de la letra que nos han dado**

**# print("Introduzca ISBN")**

**# isbn = input()**

**# respuesta = libreria24validaciones.validarISBN(isbn) #???**

**# if (respuesta == True):**

**# print("El ISBN esta BIEN")**

**# else:**

**# print("El isbn no es correcto")**

**print("Fin del programa Main")**

**UTILIZACION DE LIBRERIAS/PAQUETES EXTERNOS**

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

Ahora mismo, si no tenemos nuestro fichero de librerías, no podemos utilizarlo en casa.

Existe la posibilidad de hacer un Package de Python con nuestras librerías.

Pero el problema sigue siendo el mismo, es decir, si no tengo ese paquete accesible no tengo las librerías para poder trabajar.

Existe un repositorio de librerías para todo Python.

En dicho repositorio están todos los tipos de librerías que se nos ocurran, desde librerías para Web Scraping como librerias para leer bases de datos.

Cualquiera puede subir sus paquetes, publicarlos y acceder a ellos desde cualquier lugar con Python.

Para instalar estas librerías necesitamos Python 3.11 o superior.

Para poder utilizar estas librerías necesitamos un comando llamado **pip**

Dicho comando nos permite agregar todas las librerias de paquetes a nuestro equipo.

Tenemos la página web accesible:

<https://pypi.org/project/pip/>

El comando para instalar librerías es el siguiente:

**pip install NOMBRE LIBRERIA** (Casa)

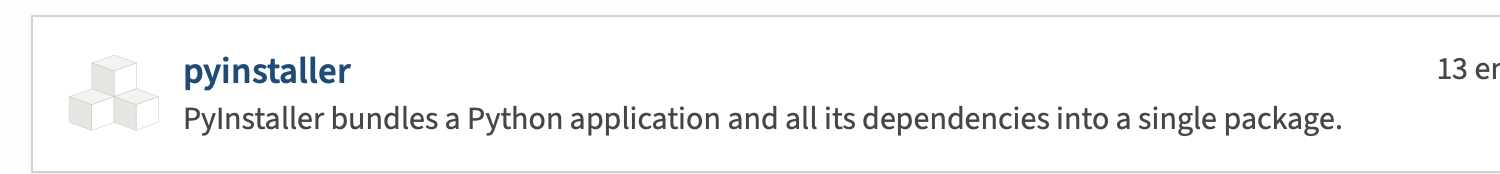
En el SO MacOS se utiliza **pip3**

Vamos a crear un instalador de nuestra aplicación, de forma que lo podremos ejecutar sin necesidad de tener Python ni compilador, ni VS Code.

En Windows, esto nos genera un .exe, en macOS nos genera un PKG

**Nota:** No importa dónde pongamos el comando de

**pip install**, las librerias se instalan en el SO.



Abrimos nuestro Terminal

**pip3 install pyinstaller**

Esta librería lo que hace es crear un ejecutable en el SO.

Para probarla, vamos a entrar en la carpeta de nuestro proyecto y vamos a seleccionar cualquier Python que tengamos que no tenga dependencias (librerías)

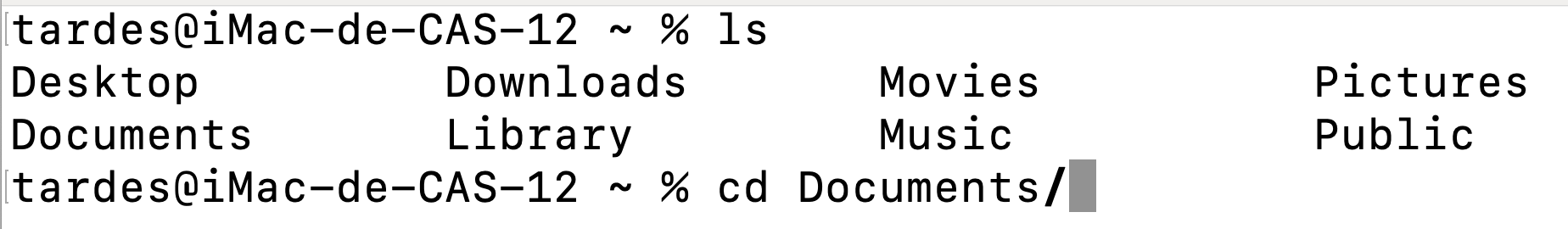
Collatz, dia nacimiento.

Desde el terminal, vamos entrando a nuestra carpeta

Con el comando **ls** vemos las carpetas

**cd NombreCarpeta**

Si pulsamos el tabulador, según escribimos, nos autocompleta



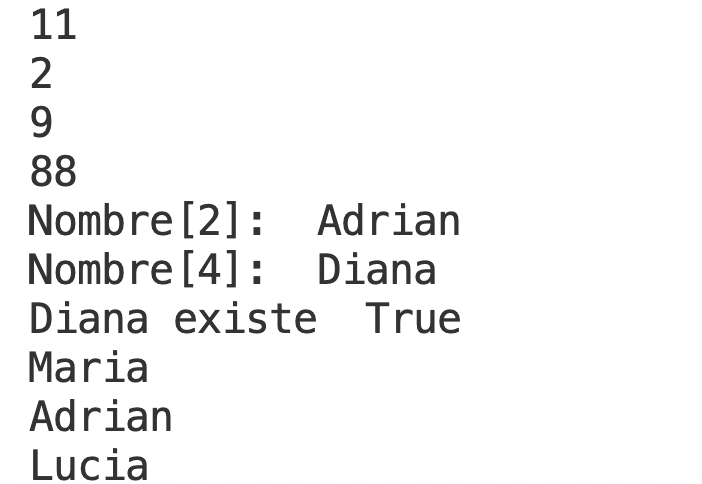
Para poder crear un ejecutable con Python, debemos escribir el siguiente comando:

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

Ya tengo un ejecutable sin necesidad de tener VS Code.

Creamos un nuevo programa llamado

**python25listastuplas.py**



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

**listaNumeros = [3,5,7,11,2,9,88]**

**#PODEMOS DIBUJAR DIRECTAMENTE LAS LISTAS**

**#Podemos ordenar de forma ASCENDENTE**

**#listaNumeros.sort()**

**#ORDEN DESCENDENTE**

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

**#print(listaNumeros)**

**#Podemos recorrer todos los elementos del**

**#conjunto 1 a 1 con un for (Todo empieza en CERO)**

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

**print(listaNumeros[i])**

**#Listas pueden ser de cualquier tipo**

**listaNombres = ["Ana", "Maria", "Adrian", "Lucia", "Diana", "Adrian"]**

**#Accedemos a los nombres por su índice**

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

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

**#append añade un nuevo elemento al final de la lista**

**listaNombres.append("El nuevo")**

**#Podemos insertar un nuevo elemento en una posicion**

**listaNombres.insert(1, "Infiltrado")**

**#El metodo remove() elimina el primer elemento que se encuentre**

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

**#Podemos eliminar por indice/posicion**

**listaNombres.pop(6)**

**#podemos borrar rangos**

**del listaNombres[0:2]**

**#Podemos preguntar por elementos dentro del conjunto**

**respuesta = "Diana" in listaNombres**

**print("Diana existe ", respuesta)**

**#Podemos recorrer cada elemento**

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

**print(listaNombres[i])**

**#Podemos borrar toda la lista**

**listaNombres.clear()**

**print(listaNombres)**

**print("Tuplas")**

**#Elementos que son estaticos**

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

**print("tupla[1], ", tupla[1])**

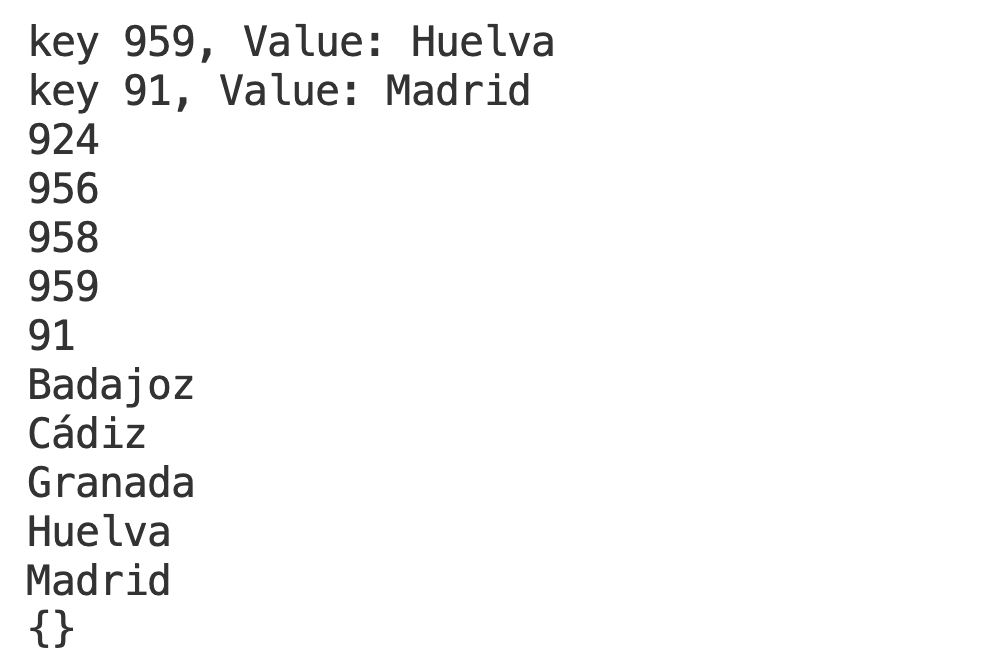
**#La tupla no se puede modificar**

**#tupla[1] = "Vainilla"**

**print(tupla)**

Creamos un nuevo programa llamado

**python26diccionarios.py**



**print("Diccionarios")**

**#Creamos el diccionario**

**provincias = *dict*()**

**#añadimos los elementos**

**provincias = {**

**924 : "Badajoz"**

**, 956: "Cádiz"**

**, 958 : "Granada"**

**, 959 : "Huelva"**

**, 91 : "Madrid"**

**}**

**#Recuperar el value de un elemento por su KEY**

**#Si no existe, no pasa nada, devuelve None**

**print(provincias.get(959))**

**#Recorrer cada Key, Value mediante items**

**for clave, valor in provincias.items():**

**print("key " + *str*(clave) + ", Value: " + valor)**

**#Podemos recorrer o solo claves (keys)**

**#o solamente valores (values)**

**for clave in provincias.keys():**

**print(clave)**

**for valor in provincias.values():**

**print(valor)**

**#Podemos añadir nuevos elementos key:value**

**provincias.setdefault(925, "Toledo")**

**#No podemos repetir una Key**

**provincias.setdefault(925, "Toledo")**

**#Podemos repetir el value**

**provincias.setdefault(933, "Toledo")**

**#Eliminamos un elemento por su Key**

**#La clave debe existir o tendremos error**

**provincias.pop(924)**

**#Eliminar toda el diccionario**

**provincias.clear()**

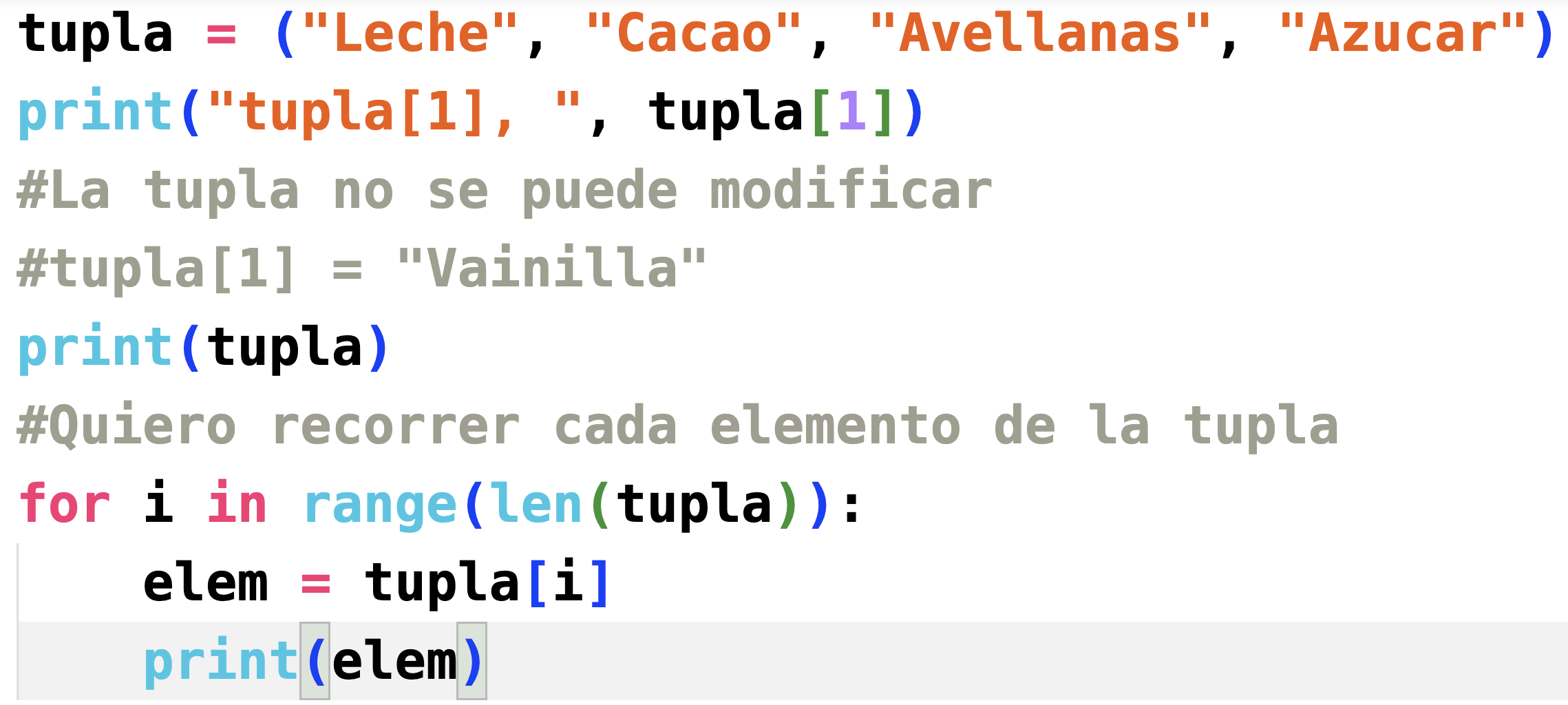
**print(provincias)**

Cuando hablamos de objetos conjuntos, existe un tercer tipo de bucle llamado de referencia.

En lugar de recorrer por una condición o por un contador, lo que realiza es recorre un conjunto y nos ofrece cada elemento del conjunto.

En este bucle, estamos recorriendo con un contador y estamos almacenando cada elemento dentro de una variable llamada **elem.**

La variable **elem** es una variable de referencia



Los bucles de refencia se utilizan cuando NO me importa el contador, solo el contenido



Creamos un nuevo programa llamado

**python27listanombres.py**

Quiero que el programa pida 5 nombres al usuario.

Dichos nombres, los mostraremos al final, después de haberlos pedido.

Si el usuario introduce un nombre repetido, le pedimos otro nombre de nuevo, es decir, quiero 5 nombres sin repetir.

**Versión 1**

**print("Nombres en listas")**

**#Declarar una lista vacia**

**lista = []**

**#Entramos en un bucle infinito hasta tener**

**#los 5 nombres**

**while (len(lista) != 5):**

**print("Dime un nombre")**

**nombre = input()**

**if (nombre in lista):**

**print("Nombre repetido")**

**else:**

**lista.append(nombre)**

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

**for n in lista:**

**print(n)**

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

**Versión 2**

Me gustaría que no importe cómo hemos escrito el nombre, no importa si está en mayúsculas o minúsculas



**print("Nombres en listas")**

**#Declarar una lista vacia**

**lista = []**

**#Entramos en un bucle infinito hasta tener**

**#los 5 nombres**

**contador = 1**

**while (len(lista) != 5):**

**print("Dime el nombre ", contador)**

**nombre = input()**

**print("Nombres en lista: ", len(lista))**

**#Debemos recorrer cada nombre dentro de nuestra lista**

**if (len(lista) == 0):**

**#Todavia no tenemos nombres**

**lista.append(nombre)**

**contador = contador + 1**

**else:**

**#recorremos todos los nombres**

**#hasta el final, no añadimos**

**existe = False**

**for name in lista:**

**print("Estoy en bucle")**

**if (name.upper() == nombre.upper()):**

**#Esta repetido**

**existe = True**

**#Podemos salir de un bucle a la fuerza**

**#si lo deseamos**

**break**

**if (existe == False):**

**lista.append(nombre)**

**contador = contador + 1**

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

**for n in lista:**

**print(n)**

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

Vamos a realizar otro ejemplo llamado

**python28listanumeros.py**

Tendremos un programa que nos irá pidiendo números hasta que el usuario introduzca el valor -1

Al final de la petición de números, necesitamos mostrar los

Siguientes datos:

\* La suma total de los números

\* Números introducidos: 9

\* Números pares: 3

\* Números impares: 6

\* Suma Pares: 68

\* Suma impares: 99

Mostrar los pares, los impares y todos los números introducidos

**print("Ejemplo listas")**

**numero = 0**

**suma = 0**

**sumaPares = 0**

**sumaImpares = 0**

**listaNumeros = []**

**listaPares = []**

**listaImpares = []**

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

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

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

**suma = suma + numero**

**listaNumeros.append(numero)**

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

**listaPares.append(numero)**

**sumaPares = sumaPares + numero**

**else:**

**listaImpares.append(numero)**

**sumaImpares = sumaImpares + numero**

**print("Datos de los numeros")**

**print("Numeros introducidos: ", len(listaNumeros))**

**print("Numeros Pares: ", len(listaPares))**

**print("Numeros impares: ", len(listaImpares))**

**print("Suma pares ", sumaPares)**

**print("Suma impares ", sumaImpares)**

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

**#Dibujar cada una de las listas**

**print("------Todos los numeros-----")**

**for num in listaNumeros:**

**print(num)**

**print("------Numeros impares-------")**

**for num in listaImpares:**

**print(num)**

**print("----Numeros Pares-------")**

**for num in listaPares:**

**print(num)**

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

RETO

1) Necesito un programa para averiguar si un número es narcisita

Un número narcisista es aquel que la suma de sus potencias elevada a su longitud nos devuelve el mismo número

Ejemplo: 153

1x1x1= 1

5x5x5=125

3x3x3=27

1 + 125+27 = 153

**import math**

**print("Numero narcista")**

**print("Introduce un numero")**

**numeroTexto = input()**

**longitud = len(numeroTexto)**

**suma = 0**

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

**letraNumero = numeroTexto[i]**

**numero = *int*(letraNumero)**

**potencia = math.pow(numero, longitud)**

**suma = suma + potencia**

**#Comprobamos los valores**

**if (suma == *int*(numeroTexto)):**

**print("Narcisista")**

**else:**

**print("No es narcisista")**

**print("fin de programa")**

2) Llevar la lógica a un método def en una librería

**import math**

***def* numeroNarcisista(*narcisista*):**

**numeroTexto = *str*(narcisista)**

**longitud = len(numeroTexto)**

**suma = 0**

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

**letraNumero = numeroTexto[i]**

**numero = *int*(letraNumero)**

**potencia = math.pow(numero, longitud)**

**suma = suma + potencia**

**#Comprobamos los valores**

**if (suma == narcisista):**

**return True**

**else:**

**return False**

3) Crear un programa en el que el usuario introduce un rango y le mostramos los números narcisistas en ese rango con nuestro método

**import libreria29narcisista**

**print("Numero narcista")**

**print("Introduce un numero iniciar")**

**inicial = *int*(input())**

**print("Introduzca numero final")**

**final = *int*(input())**

**for inicial in range(inicial, final):**

**if (libreria29narcisista.numeroNarcisista(inicial) == True):**

**print(inicial)**

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

Necesito poder almacenar la información de temperaturas de un mes

Un mes está compuesto por un Nombre, una temperatura máxima y una temperatura mínima.

Me gustaría saber la temperatura media del mes.

Creamos una nueva clase llamada **class32mes.py**

***class* Mes:**

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

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

***self*.*temperaturaMinima* = 0**

***self*.*temperaturaMaxima* = 0**

**#Necesitamos un metodo para calcular la media de un mes**

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

**media = (*self*.*temperaturaMaxima* + *self*.*temperaturaMinima*) / 2**

**return media**

A continuación, creamos un programa para probar esta funcionalidad

**python32pruebameses.py**

**from class32mes import Mes**

**print("Trabajando con clases")**

**enero = Mes()**

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

**enero.*temperaturaMaxima* = 9**

**enero.*temperaturaMinima* = -2**

**print("Media Enero ", enero.getTemperaturaMedia())**

**mes2 = Mes()**

**mes2.*nombre* = "Febrero"**

**mes2.*temperaturaMaxima* = 12**

**mes2.*temperaturaMinima* = 4**

**print("Media febrero ", mes2.getTemperaturaMedia())**

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

El siguiente paso que quiero hacer es pedir al usuario información sobre los 3 primeros meses.

Al finalizar de introducir los datos, mostraremos todos los meses con su información

**from class32mes import Mes**

**print("Trabajando con clases")**

**listaMeses = []**

**for i in range(3):**

**#Aqui creamos un nuevo mes**

**mes = Mes()**

**print("Introduzca el mes ", (i + 1))**

**mes.*nombre* = input()**

**print("Temperatura máxima")**

**mes.*temperaturaMaxima* = *int*(input())**

**print("Temperatura mínima: ")**

**mes.*temperaturaMinima* = *int*(input())**

**listaMeses.append(mes)**

**#Recorremos los meses que hemos almacenado**

**for dato in listaMeses:**

**print(dato.*nombre* + ", Maxima " + *str*(dato.*temperaturaMaxima*))**

**print("Minima " + *str*(dato.*temperaturaMinima*))**

**print("Media ", dato.getTemperaturaMedia())**

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

Quiero tener la información de varias Mascotas

Una mascota está compuesta por un Nombre, una raza,

Año de nacimiento y año de adopción.

Necesito saber el número de años adoptada desde el nacimiento.

1) Creamos la clase

2) Utilizamos la clase simple

3) Pedimos tres mascotas, las guardamos en «algun sitio» y las mostramos

**INSTALACION BBDD**

Vamos a utilizar la base de datos Oracle para acceso a datos, de hecho, utilizaremos otras también para probar.

En Mac, la instalación de Oracle no es nativa, es decir, si queremos trabajar con la base de datos Oracle (sin pagar), debemos utilizar **docker**

CASA:

Opción 1: Docker, mismos pasos e instalamos el IDE del Castor o del SQL Developer

Opción 2:

En Windows, ningún problema, luego os dejo el link en

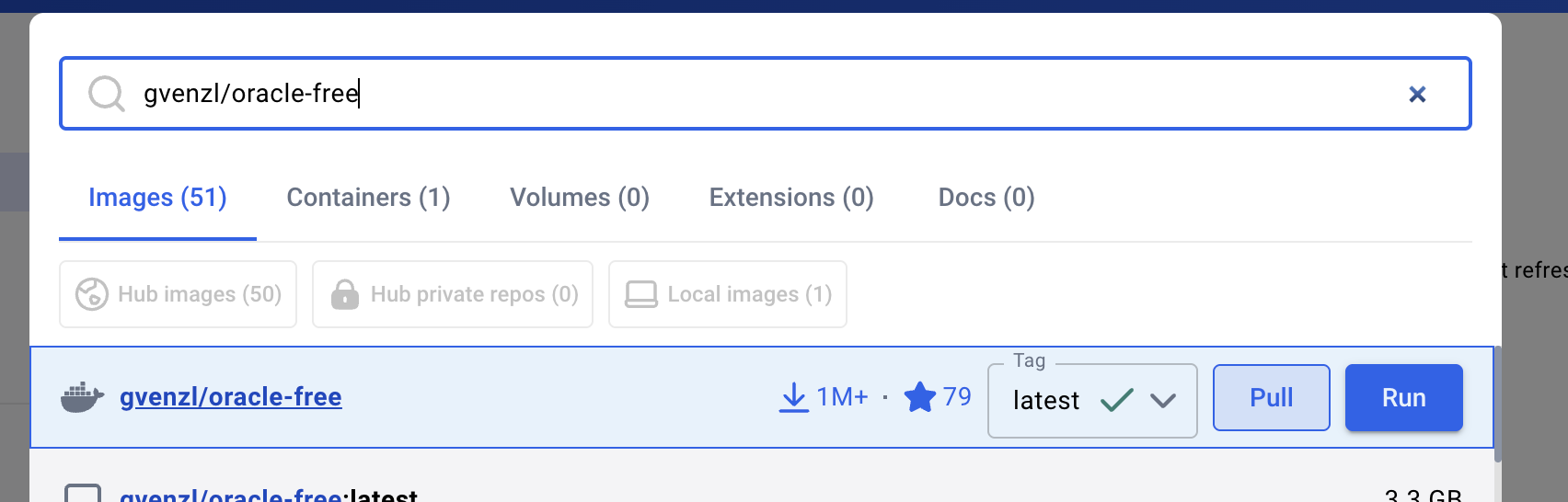
**Github** para que podáis intalar Oracle sin Docker.

Descargamos Docker

<https://www.docker.com>

Las imágenes podemos lanzarlas y comunicar con ellas cuando deseemos, ya sea desde Command Line o desde la parte Visual

Esta imagen de Docker, necesita de un User y Pass para trabajar con ella



Lanzamos nuestro container con el siguiente comando en Terminal:

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

En images, tendremos nuestra imagen montada con los siguientes datos de conexión:

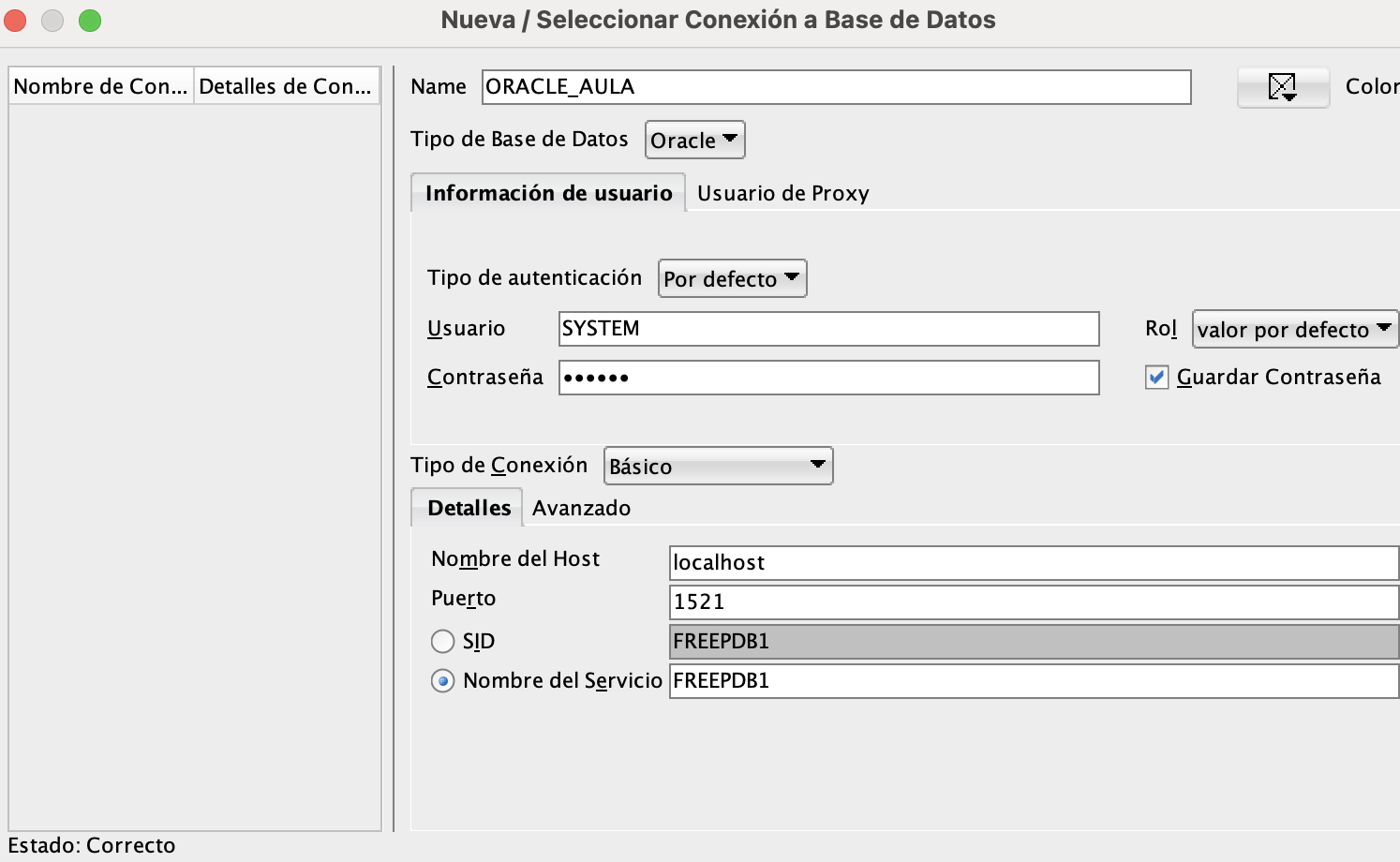
Host: LOCALHOST

User: **SYSTEM**

Password: **oracle**

Port: 1521

Nombre del Servicio: **FREEPDB1**



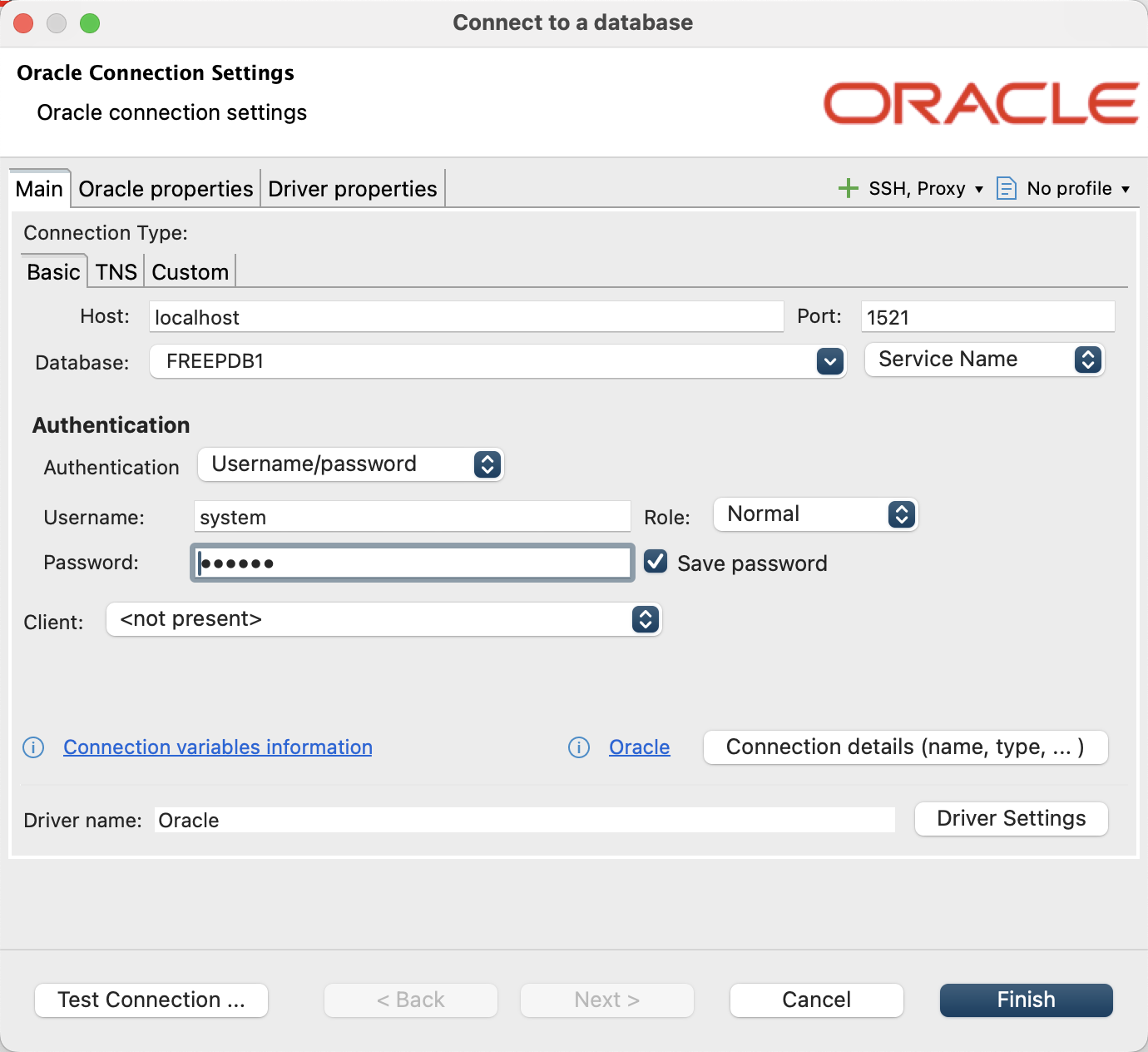
Cada día que deseemos Oracle, simplemente, dentro de Docker, Iniciamos la imagen.

**IDE ORACLE**

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

Para casa, tenemos otro que ocupa mucho menos que es Dbeaver. La ventaja está en que podemos atacar a múltiples bbdd con este IDE

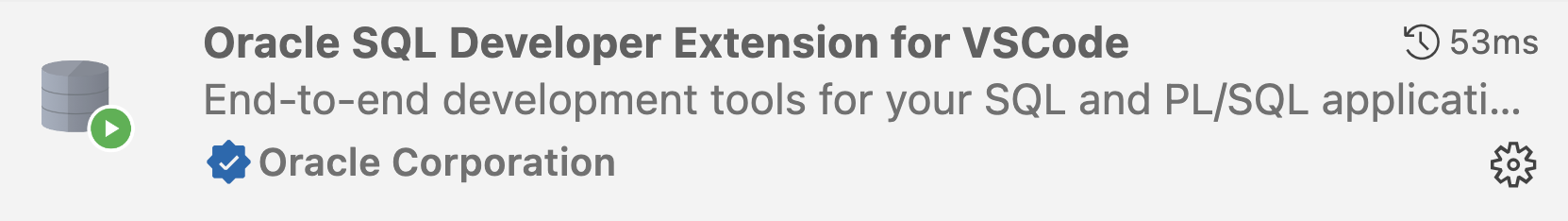
<https://dbeaver.io>

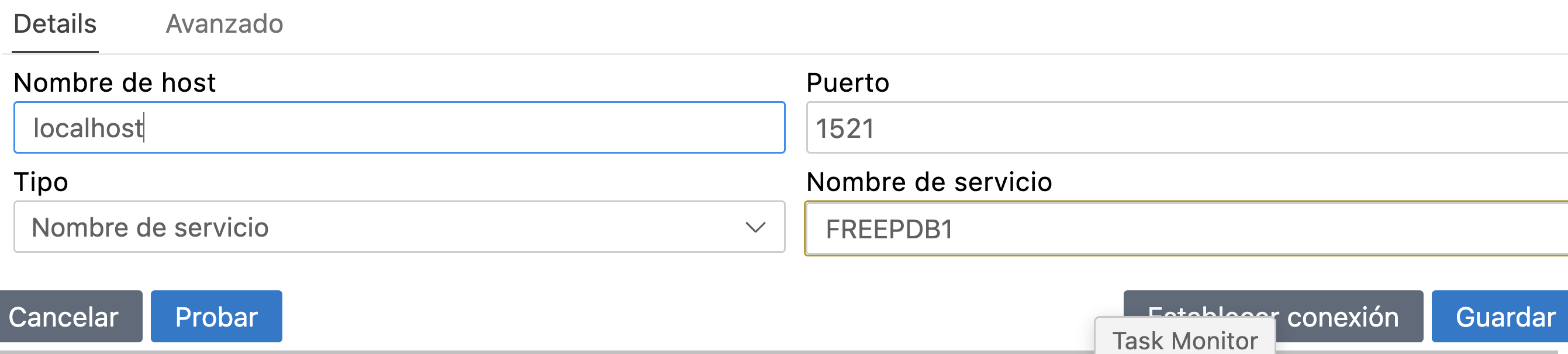


También podemos utilizar como IDE nuestro VS Code.

Debemos instalar la siguiente extensión:

**Nota:** Esta extensión es solo para Oracle.

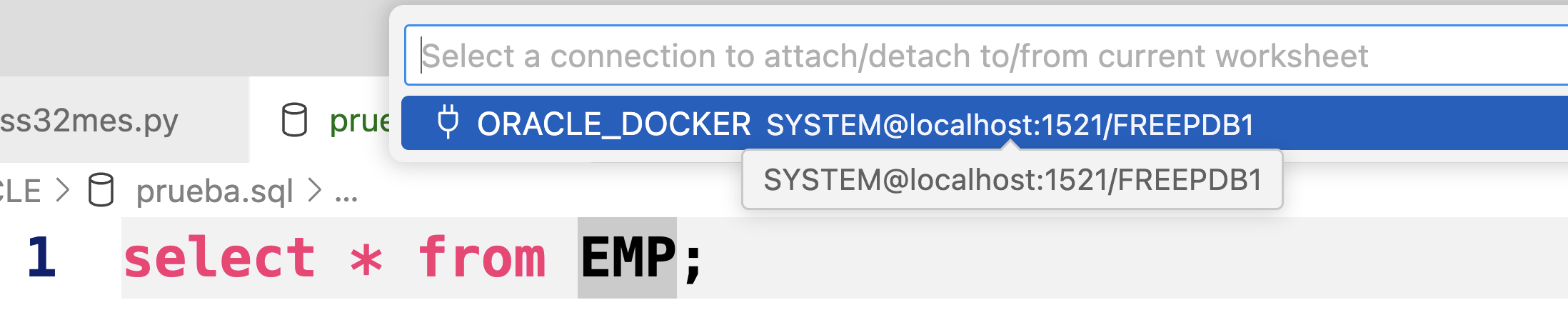




Para configurar en la parte de la izquierda tenemos como una especie de pila.



Para ejecutar consultas, simplemente creamos, en nuestro proyecto una fichero con extensión **.sql**



Tendremos una base de datos de pruebas e iremos viendo diferentes consultas.

Por último, vamos a incluir nuestras tablas dentro de

Oracle.

Todo lo que vaya contando, lo tenéis ya documentado.

**Ansi SQL Server**

Es un lenguaje estandar de bases de datos relacionales.

Desde 1992 todas las bases de datos deben cumplir esta norma si quieren tener el certificado ISO.

Consultas selección: Sirven para recuperar datos

\* Simples

\* Agrupación

\* Combinación

\* Subconsultas

Consultas de acción: Sirven para actualizar los datos

\* Insert

\* Update

\* Delete

--PODEMOS TENER VARIAS CONSULTAS EN ESTE SCRIPT

--SOLAMENTE SE EJECUTA EN LA QUE ESTEMOS POSICIONADOS (CONTROL + ENTER)

--NO EXISTE DIFERENCIA CASE SENSITIVE

--EL \* SE UTILIZA PARA INDICAR TODAS LAS COLUMNAS DE UNA TABLA

--select COLUMNA1, COLUMNA2 from TABLA;

select \* from DEPT;

select DNOMBRE from DEPT;

--EXISTEN CONSULTAS QUE SON MAS OPTIMAS QUE OTRAS Y DEVUELVEN LO MISMO

select DEPT\_NO, DNOMBRE, LOC from DEPT;

--ORDENAR LOS DATOS: order by CAMPO

select \* from EMP order by SALARIO desc;

select \* from EMP order by OFICIO desc;

--FILTROS DE CONSULTAS

--permiten filtrar registros por columnas en sus datos, utilizan el operador where

--Si el dato a filtrar NO es un número, se utilizan las comillas simples '

--operadores condicionales

/\*

= IGUAL

<> DISTINTO

> MAYOR

>= MAYOR O IGUAL

< MENOR

<= MENOR O IGUAL

\*/

--mostrar los empleados del departamento 10

--where CAMPO = VALOR

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

--mostrar todos los empleados con oficio director

--Oracle diferencian en sus datos STRING

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

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

--mostrar los empleados que sean distintos al departamento 10

select \* from EMP where DEPT\_NO <> 10;

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

--OPERADORES RELACIONALES

--permiten filtrar por mas de una condicion, solamente podemos tener un where

--and: Todas las condiciones se deben cumplir

--or: Devuelve registros de cada condición

--not: Negación de una condición.

--Mostrar todos los empleados con oficio director y un salario mayor a 300.000

select \* from EMP where OFICIO='DIRECTOR'

and SALARIO > 300000;

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

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

--Mostrar todos los empleados que NO sean del departamento 20

select \* from EMP where NOT DEPT\_NO=20; --negacion nunca

select \* from EMP where DEPT\_NO <> 20;

--Existen otra serie de operadores de comparación además de los básicos

--between busca entre dos rangos inclusive

--Mostrar todos los empleados que tengan un salario entre 123500 y 318500

select \* from EMP where SALARIO between 123500 and 318500;

select \* from EMP where SALARIO >= 123500 and SALARIO <= 318500;

--Operador IN permite buscar entre diferentes valores de un mismo campo

--Mostrar todos los empleados del departamento 10 y 20 y del 30

--y del 40, 50,66,88

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

or DEPT\_NO=40 or DEPT\_NO=50 or DEPT\_NO=66 or DEPT\_NO=88;

select \* from EMP where DEPT\_NO in (10, 20, 30, 40, 50, 66,88);

--Operador NOT IN que busca entre diferentes valores de un mismo campo y

--devuelve los que no correspondan

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

select \* from EMP where NOT DEPT\_NO in (10, 20); --ESTO ES UNA NEGACION, NO EFICIENTE

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

--Tenemos un operador que nos permite hacer búsquedas dentro de Textos.

--Es el operador like y utiliza caracteres especiales para las busquedas:

-- ? Busca si el caracter es un número

-- \_ Cualquier unico caracter

-- % Cualquier caracter y cualquier tamaño

--Mostrar todos los empleados cuyo apellido comienza en s

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

--Mostrar todos los empleados cuyo apellido comienza en s y finalicen en a

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

--Mostrar todos los empleados cuyo apellido tenga 4 letras

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

select \* from EMP where APELLIDO like '\_a%';

--Clausula distinct elimina datos repetidos de la consulta

--Se suele utilizar con pocos campos

select distinct OFICIO from EMP;

--Campos calculados. Son columnas que no existen en la tabla y que

--podemos generar con una consulta.

--Los campos calculados siempre deben tener un ALIAS

select APELLIDO, OFICIO, SALARIO + COMISION as TOTAL from EMP;

--No podemos utilizar where con campos calculados (mentira).

--where filtra sobre la tabla

--Si queremos filtrar sobre un campo calculado, debemos escribir el

--calculo de nuevo para el where

select APELLIDO, OFICIO, SALARIO + COMISION as TOTAL from EMP

where (SALARIO + COMISION) >= 300000;

--Si podemos utilizar order by con campos calculados.

--order by ordena sobre el cursor

select APELLIDO, OFICIO, SALARIO + COMISION as TOTAL from EMP order by TOTAL;

**CONSULTAS DE AGRUPACION**

Este tipo de consultas NUNCA devuelven los registros de la tabla.

Sirven para recuperar resúmenes de los registros de la tabla.

Por ejemplo, saber el número de registros de la tabla.

Averiguar el máximo valor de un dato de la tabla.

Se llaman consultas de agrupación porque podemos agrupar los resultados respecto a uno o varios campos/columnas.

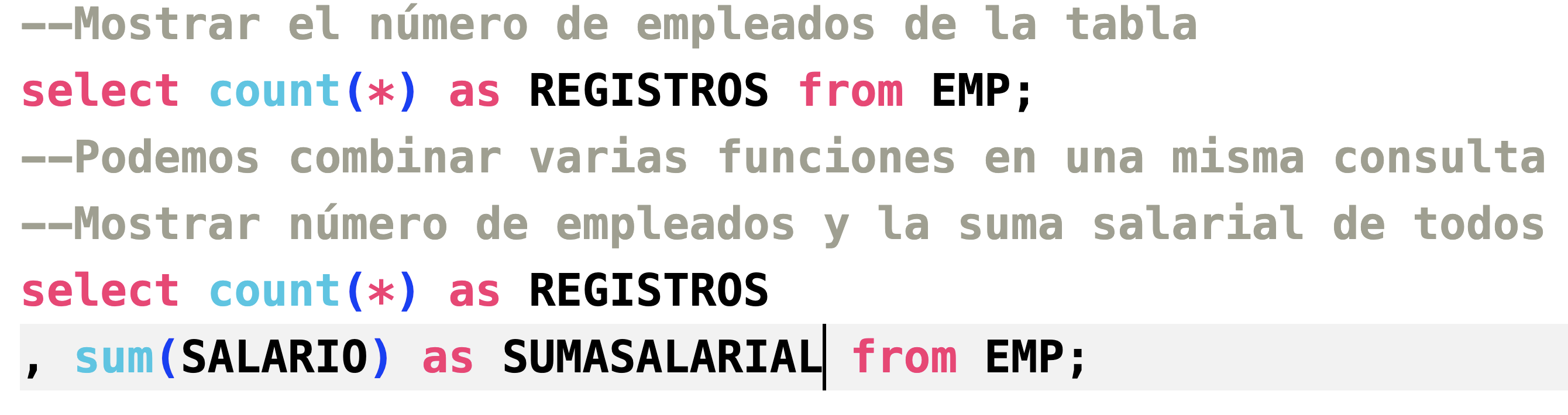
Por ejemplo, saber el número de registros de los empleados por cada departamento.

Averiguar el máximo salario por cada oficio.

Utilizan funciones de agrupación para recuperar los datos.

* COUNT(\*): Cuenta el número de registros, contando los datos null.
* COUNT(COLUMNA): Cuenta el número de registros, sin contar null.
* SUM(COLUMNA NUMERO): Devuelve la suma total de una columna.
* AVG(COLUMNA NUMERO): Devuelve la media de una columna.
* MAX(COLUMNA): Devuelve el valor máximo
* MIN(COLUMNA): Devuelve el valor mínimo

**Nota:** Las funciones de agrupación siempre deben llevar ALIAS.

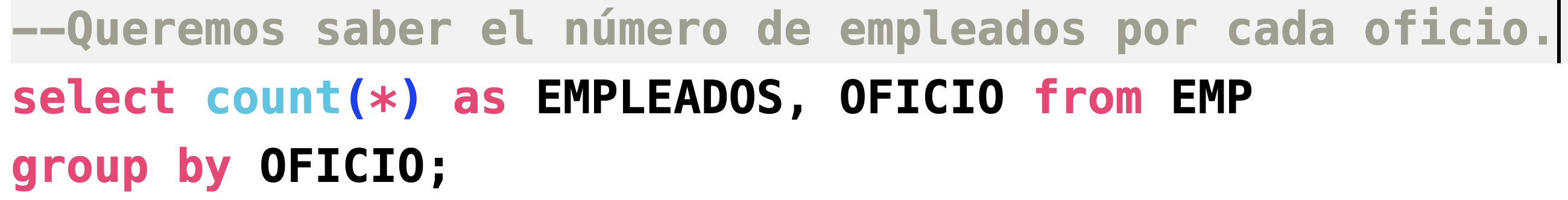


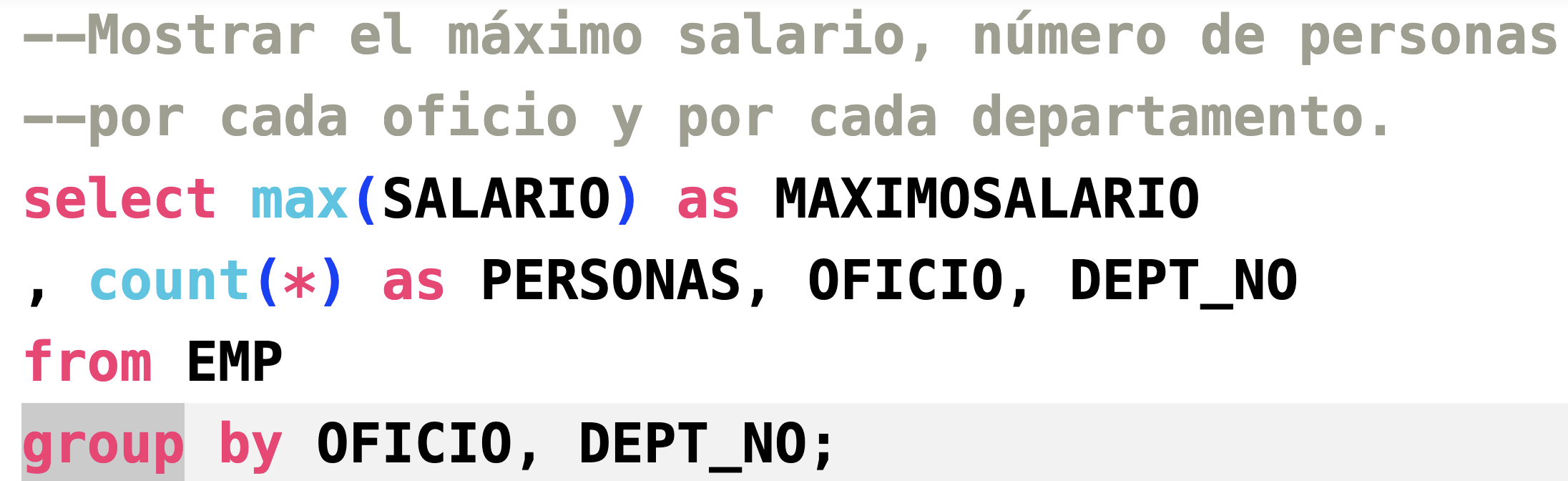
Lo que nos interesa en estas consultas es la agrupación por algún otro dato.

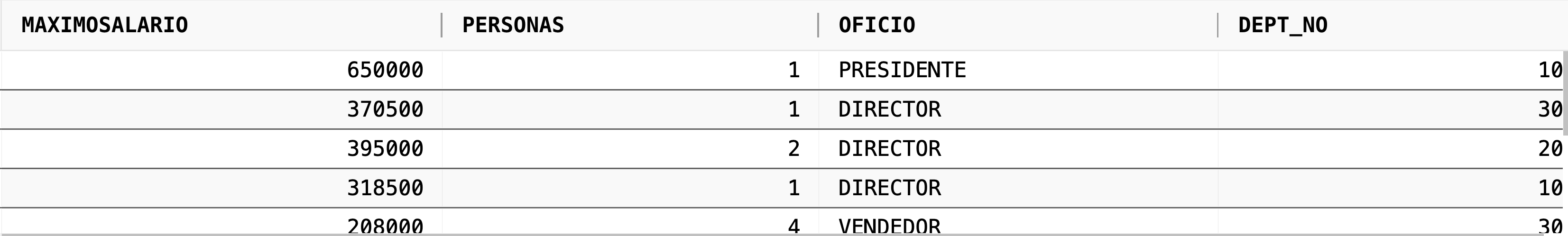
Por ejemplo, saber cuantas personas tenemos por cada OFICIO.

1. La agrupación se realiza en el SELECT con la columna o columnas que deseemos, además de la función o funciones de agrupación.
2. Después del FROM, indicaremos los campos por los que estamos agrupando mediante **group by**

**Truco:** Las columnas que NO sean una función que pongamos en el select, deben estar en el group by.







Por supuesto, podemos filtrar los datos que deseamos agrupar.

Tenemos dos opciones para filtrar:

* **where:** Filtra por los datos de la tabla. Se escribe **ANTES DEL GROUP BY**
* **having:** Filtra por las funciones de agrupación. Se escribe **DESPUES DEL GROUP BY**

Estructura:

select CAMPO1, CAMPO2 from TABLA;

Con Where

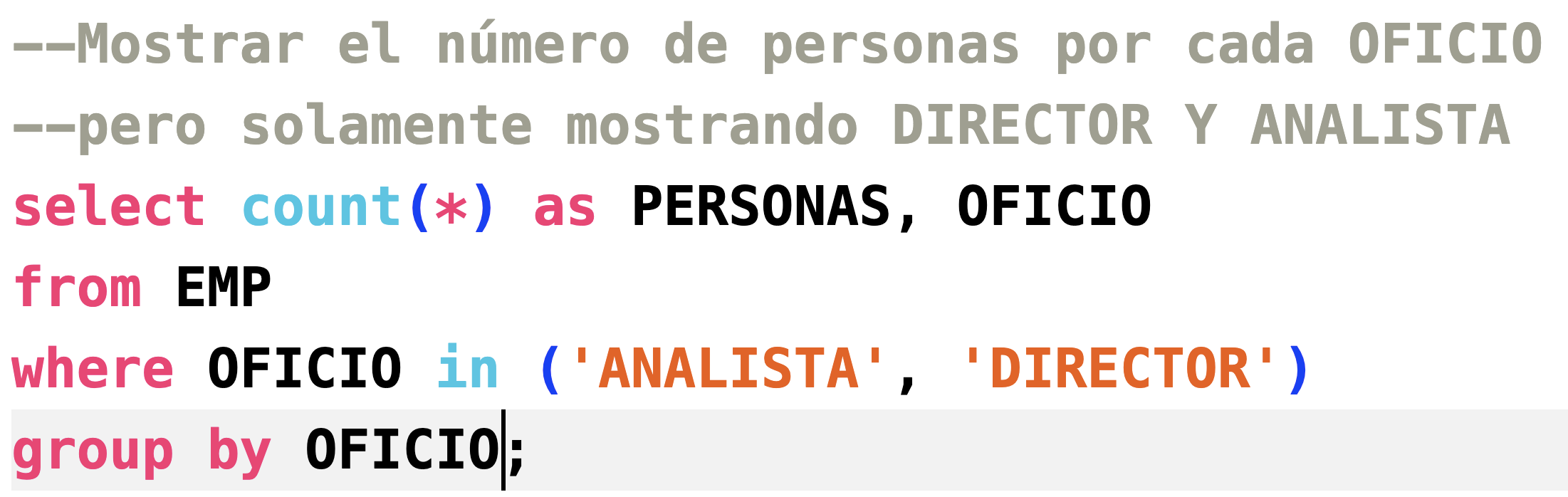
select CAMPO1, CAMPO2 from TABLA where CONDICION;

Con group by

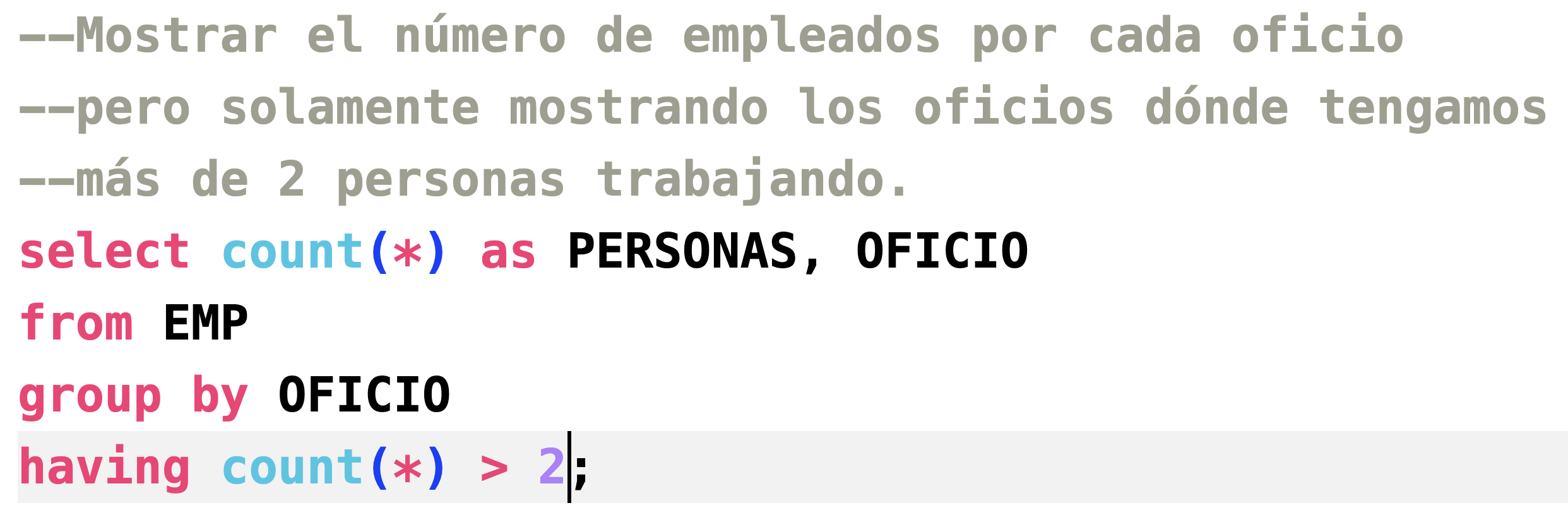
select CAMPO1, CAMPO2 from TABLA group by CAMPO;

Con Where y group by

select CAMPO1, CAMPO2 from TABLA where CONDICION group by CAMPO;



Ejemplo con HAVING





Por supuesto, podemos combinar los dos, todo depende de la consulta



Podemos combinar where con having, por supuesto

