PYTHON BASICO

Como vamos a trabajar

Vamos a utilizar un entorno de trabajo colaborativo para que tengáis acceso a todo, mis códigos y que también tengáis vuestros códigos.

Lo que haremos será trabajar con **Github**.

1. Instalar Python

Necesitamos un IDE para trabajar. El IDE es **Visual Studio Code** que sirve para programar con Python

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

El contenido generado por IA puede ser incorrecto.

Debemos separa los conceptos:

1. Jupyter es un cuaderno de trabajo Python. Sirve para análisis de datos
2. Python básico se mueve en cualquier script.

Todo esto está reunido dentro de cualquier entorno.

Para trabajar con Jupyter, necesitamos un entorno.

Para trabajar con Script, necesitamos un entorno.

Para trabajar con Fast Api o Django, necesitamos un entorno.

La diferencia es que mediante Visual Studio Code, podemos utilizar TODO.

Mediante extensiones, podemos hacer que nuestro entorno funcione sin problemas dentro de VS Code.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Una vez que tenemos esto, ya se ha ejecutado un elemento para poder crear aplicaciones de Python

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

Simplemente, cualquier fichero que tengamos se puede ejecutar mediante VS Code sin necesidad de nada más.

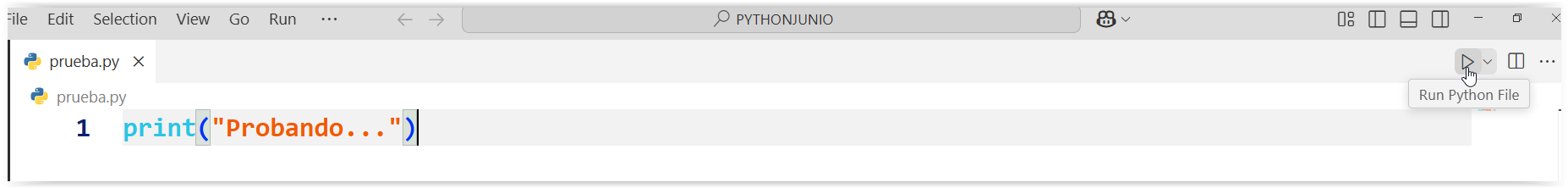
Para trabajar, simplemente tendremos una carpeta con nuestros ficheros Python.

Dependiendo del proyecto/framework se generan más carpetas. Pero en este tipo básico, simplemente necesitamos una carpeta en nuestro ordenador y dentro iremos creando todas las clases y características

PYTHONJUNIO

Interfaz de usuario gráfica, Aplicación, Tabla

El contenido generado por IA puede ser incorrecto.



Python es un lenguaje de programación. Contiene programación orientada a objetos y se comunica con múltiples elementos, por ejemplo, Web, IoT, Api, Análisis de datos, Big Data…

**Importante:**

* **Diferencia mayúsculas de minúsculas**
* **Las tabulaciones son básicas, delimitan el código del programa**
* **Las variables (mentira) no tienen tipado, simplemente le asignamos un valor.**

Las variables se declaran en el código

numero = 14

texto = “Soy un texto”

**print** en nuestro entorno será el que nos servirá para visualizar que sucede. Dependiendo del Framework que utilicemos, no se utiliza print

Dependiendo del tipo de variable, podremos sumar o restar o dividir o podremos buscar una letra o convertir a mayúsculas

Para trabajar y ser ordenados, iremos nombrando a nuestros ficheros de una forma determinada. En principio no importa el nombre del fichero, pero normalmente, en Python se declaran los programas en minúsculas.

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

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

**numero = 19**

**texto = "Hola mundo"**

**#Comentarios en Python**

**# Para comentar y descomentar código de una sola vez en VS Code.ç**

**# Comentar:  CONTROL + K + C**

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

**print(numero)**

**print(texto)**

**# Tenemos la posibilidad de concatenar +: Verifica el tipado**

**# print("numero " + numero) # error**

**# Con coma no verifica tipado**

**print("numero ", numero)**

**print("texto " + texto)**

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

**#importar el tipado: Cada variable irá entre llaves {variable}**

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

**#Dentro de las variables tenemos funciones de conversión lo que permiten convertir**

**#entre un tipo y otro, por ejemplo:**

**#str(variable): Convierte un valor a string**

**#float(variable): Convierte variable a tipo decimal**

**#int(variable): Convierte variable a entero**

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

CONDICIONALES

Un condicional dentro de cualquier lenguaje implica que un programa no será lineal.

Para poder utilizar condicionales/preguntas en nuestro programa se utilizan las TABULACIONES para delimitar si la pregunta es correcta o no es correcta.

Al hablar de condiciones, estamos hablando de IF

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Para poder comparar, necesitamos la sintaxis de los operadores de comparación de Python.

> Mayor

< Menor

>= Mayor o igual

<= Menor o igual

== Igual

!= Distinto

Vamos a realizar un simple programa en el que indicaremos si un número es positivo o negativo o cero.

Pediremos el número al usuario mediante una función de Python llamada **input()**

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

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

ELSE

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

ELSE IF ANIDADO

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Si tiene que ver con la misma pregunta, lo que podemos utilizar es la sentencia **elif**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**DEPURACION DE CODIGO**

La depuración de código nos permite visualizar las variables dinámicamente en nuestro programa o nos permite averiguar que ha sucedido.

En cualquier programa tenemos tres tipos de errores:

1. **Errores de compilación**: Algo he escrito mal y el programa no funciona.
2. **Errores en ejecución**:El programa compila y ejecuta, pero se me ha escapado algún detalle de comprobación. Por ejemplo, en este ejemplo anterior, comprobar que ha introducido números.
3. **Errores lógicos**: Simplemente mi programa funciona, no da ningún error, pero no hace lo que quiero. Debemos averiguar qué sucede

Para depurar en Python con VS Code, simplemente debemos agregar un Punto de interrupción dentro de nuestro programa en la parte de la izquierda de los números.

Texto

El contenido generado por IA puede ser incorrecto.

Para ejecutar el programa en modo depuración, tenemos un icono de un bicho a la izquierda de nuestro código llamado Run and Debug

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Cuando hablamos de condicionales, tenemos también los operadores relacionales:

Son dos palabras clave:

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

Para visualizar esto y tener un ejemplo de sintaxis:

Creamos una nueva App llamada **python03mayormenor.py**

Vamos a pedir tres números y mostramos el mayor, menor e intermedio de ellos.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**print("Operadores relacionales")**

**num1 = *int*(input("Número 1"))**

**num2 = *int*(input("Número 2"))**

**num3 = *int*(input("Número 3"))**

**mayor = 0**

**menor = 0**

**intermedio = 0**

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

**mayor = num1**

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

**mayor = num2**

**else:**

**mayor = num3**

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

**menor = num1**

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

**menor = num2**

**else:**

**menor = num3**

**suma = num1 + num2 + num3**

**intermedio = suma - mayor - menor**

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

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

BUCLES

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

Tenemos dos tipos de bucles:

1. **while:** Bucle condicional que debemos entrar y salir con una condición. Dentro del bucle debemos cambiar “algo” para que se cumpla la condición y el bucle finalice.

Sintaxis:

while (condición == true):

#instrucciones

condición = false

print(“Fin de programa”)

1. **for:** Bucle que nos permite repetir una secuencia N veces. La secuencia se realiza mediante un contador declarado dentro de la sintaxis del propio bucle.

Entra y sale solo, no hace bucles infinitos.

Un bucle contador for siempre comienza en CERO

Sintaxis:

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

#instrucciones

Tenemos otro tipo de sintaxis en el que podemos indicar el inicio y el final de contador

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

#instrucciones

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

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Vamos a realizar un ejemplo con la Conjetura de Collatz.

La conjetura indica lo siguiente:

Todo número entero siempre llegará a ser 1 si se cumplen las siguientes condiciones:

* Si el número es par, se divide entre 2.
* Si el número es impar, se multiplica por 3 y se suma 1

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

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

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

**#REALIZAMOS EL TIPICO DE MOSTRAR LOS NUMEROS PARES ENTRE UN RANGO**

**for i in range(2, 20):**

**#PREGUNTAMOS SI UN NUMERO ES PAR**

**#UTILIZAMOS EL OPERADOR DEL RESTO: %**

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

**print(*f*"Par: {i}")**

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

**numero = *int*(input("Introduzca numero: "))**

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

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

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

**else:**

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

**print(numero)**

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

**CLASE STRING PYTHON**

Los objetos, al ser declarados con algún tipado (todavía explícitamente no), contienen una serie de características o métodos.

En Python existen clases y, dependiendo de dichas clases, tendremos una serie de métodos que nos pueden ayudar en programación.

Por ejemplo, tenemos clases que ya vienen con nuestro compilador y no necesitamos nada más: String

Por otro lado, tendremos clases que habrá que importar y utilizarlas en el programa, por ejemplo, la clase Math.

Por último, tenemos clases que no existen en el compilador y tendremos que agregarlas a nuestro programa utilizando herramientas externas.

Tenemos una función para saber la longitud de cualquier objeto conjunto: **len()**

Un String diferencia mayúsculas de minúsculas.

texto = “Lenguaje Python”

len(texto) 🡪 15

Dentro de este conjunto, también tenemos la posibilidad de acceder a cada elemento individual (letra) mediante un 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, tenemos una funcionalidad que se llama **SLICING** que nos permite poder extraer subcadenas de un conjunto

texto = “primer python”

Tenemos dos posibilidades para obtener una subcadena de un texto a partir de sus índices:

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

texto[8:] 🡪 ython

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

texto[0:2] 🡪 pri

Vamos a realizar una aplicación para validar un mail.

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

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

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

El contenido generado por IA puede ser incorrecto.

Paco Garcia Serrano

**print("Validacion email string")**

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

**# • Que el email contenga @**

**# • Que el email contenga .**

**# • @ ni al inicio ni al final**

**# • Solamente una @ en el email**

**# • Que exista un punto después de la @**

**# • Dominio de 2 a 3 caracteres**

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

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

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

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

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

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

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

**print("Tenemos más de una @")**

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

**print("Debe existir un punto después de @")**

**else:**

**#posicion del ultimo punto**

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

**#Recuperamos el dominio a partir de dicho punto en adelante**

**#no queremos el punto incluido**

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

**longitud = len(dominio)**

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

**print(*f*"Email correcto: {email}")**

**else:**

**print("Dominio de dos a tres caracteres")**

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

Vamos a realizar un ejemplo en el que recorreremos todos los caracteres de un texto (numérico) y mostraremos la suma de cada carácter.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Creamos un programa nuevo llamado **python06numerosstring.py**

**print("Sumar números de textos")**

**textoNumeros = input("Introduzca un texto numerico: ")**

**suma = 0**

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

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

**#ALMACENAMOS CADA LETRA DE CADA POSICION**

**letra = textoNumeros[i]**

**# EXTRAEMOS CADA LETRA Y LA CONVERTIMOS A int**

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

**suma = suma + numero**

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

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

## METODOS/FUNCIONES DENTRO DEL LENGUAJE PYTHON

Una función es un método, en realidad son lo mismo.

Son herramientas que contiene el lenguaje de Python y que pueden venir dentro de una clase o no.

* Un método en una clase: La clase String tiene un método que permite buscar si una cadena comienza con un determinado Carácter/Texto: startswith()
* Método sin clase: Dichos métodos están “sueltos” y no dependen de una clase determinada. Por ejemplo: **len(texto)**

Un programa de Python, como hicimos el otro día, UTILIZA métodos y funciones de una clase o de un Programa

Estos métodos/funciones nos permiten realizar códigos NO lineales, es decir, separar ciertas características/funcionalidades del programa en partes o zonas.

Un método nos permite también poder reutilizar nuestro código, lo escribimos una vez y lo utilizamos múltiples veces.

Los métodos pueden estar en el mismo programa principal o separado en diferentes ficheros de Python.

Todo funciona mediante llamadas y declaraciones de métodos, por ejemplo, imaginemos que tenemos por ahí un método que nos valida el Email:

PROGRAM MAIN.PY

print(“Programa principal”)

print(“Introduzca un email”)

email = input()

#AQUI TODO EL CODIGO DE VALIDACION

**validarEmail()**

**metodos.validarEmail()**

#METODO DE VALIDAR UN MAIL

**validarEmail()**

{

#NUESTRO CODIGO DE VALIDACION

}

Podemos plantear nuestro programa de dos formas mediante métodos:

1. En el mismo Main donde estamos, crear un método para validar un Email
2. Podríamos tener otro fichero con métodos

**METODOS.PY**

#METODO DE VALIDAR UN MAIL

**validarEmail()**

{

#NUESTRO CODIGO DE VALIDACION

}

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

La sintaxis de los nombres de método es como en Java/JavaScript, es decir, la primera palabra en minúscula y cada palabra siguiente con la inicial en mayúscula: **nombreMetodo()**

En Python, de forma obligada, los métodos deben estar al inicio de la clase.

Por ahora, los métodos los veremos en la misma clase **main.py**.

Creamos un nuevo programa llamado **python07metodos.py**

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

El contenido generado por IA puede ser incorrecto.

**# EN UN PROGRAMA LOS METODOS DEBEN**

**# ESTAR ANTES DE LA LLAMADA, POR LO QUE**

**# SE SUELEN DECLARAR AL INICIO**

***def* primerMetodo():**

**#ESTE CODIGO JAMAS SE EJECUTARA**

**# SIN SER LLAMADO DE FORMA EXPLICITA**

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

***def* segundoMetodo():**

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

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

**#CODIGO PRINCIPAL DE NUESTRO PROGRAMA MAIN**

**print("Ejemplo métodos")**

**#LLAMADA LOS METODOS**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

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

## PARAMETROS EN LOS METODOS

Un método puede recibir parámetros, dichos parámetros NO son opcionales (por ahora), deben ser enviados al método para poder ejecutarlo en la llamada.

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

Dicha variable tendrá ámbito de método, es decir, solamente la podemos utilizar dentro del **def**

Un método puede contener múltiples parámetros, simplemente se separan mediante coma dichos parámetros.

El orden de la llamada de dichos métodos con parámetros será el orden de los parámetros

Ejemplo:

def metodoParametros(**nombre**):

print(“Su nombre es: “ + nombre)

Petición al método

metodoParametros(12)

metodoParametros(“Paco”)

def metodoParametrosMultiple(**nombre, apellidos, edad**):

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

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

Vamos a probar esto en el mismo programa que tenemos de métodos.

**#Comenta: control + k + c**

**#Descomentar: control + k + u**

**# EN UN PROGRAMA LOS METODOS DEBEN**

**# ESTAR ANTES DE LA LLAMADA, POR LO QUE**

**# SE SUELEN DECLARAR AL INICIO**

***def* primerMetodo():**

**#ESTE CODIGO JAMAS SE EJECUTARA**

**# SIN SER LLAMADO DE FORMA EXPLICITA**

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

***def* segundoMetodo():**

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

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

**print("Bienvenido/a a Python ", nombre)**

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

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

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

**#CODIGO PRINCIPAL DE NUESTRO PROGRAMA MAIN**

**print("Ejemplo métodos")**

**saludar("Paco")**

**despedirse("Paco", "Juernes")**

**#LLAMADA LOS METODOS**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

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