PYTHON BASICO

Como vamos a trabajar y requisitos necesarios

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

1. Instalar Python

Texto

El contenido generado por IA puede ser incorrecto.

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

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

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

Python se ha hecho popular por Hacking y Big Data

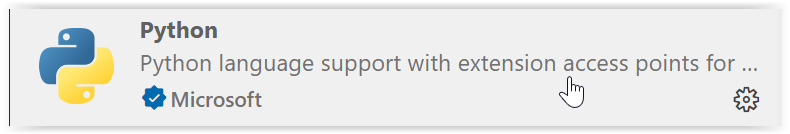
Dentro de Python tenemos multitud de características:

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

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

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

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



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

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

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

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

Abrimos esta carpeta desde Visual Studio Code

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.



**Importante en el lenguaje:**

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

Vamos a trabajar de forma consecutiva al nombrar los programas.

Llamaremos a los programas **pythonXXdescripcion.py**

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

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

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

**numero = 14**

**texto = "Primer Python"**

**#Comentarios en Python**

**#Comentar dentro de VS Code**

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

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

**print(numero)**

**print(texto)**

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

**#Verifica el tipo de dato**

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

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

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

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

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

**#importar el tipo de dato.**

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

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

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

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

**#podemos convertir mediante funciones de Python**

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

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

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

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

CONDICIONALES

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

El código no será lineal.

Los condicionales están delimitados mediante Tabulaciones en sus instrucciones.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Operadores de comparación en Python:

> Mayor

< Menor

>= Mayor o igual

<= Menor o igual

== Igual

!= Distinto

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

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

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

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Tenemos la posibilidad de incluir una condición ELSE

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

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

Texto

El contenido generado por IA puede ser incorrecto.

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

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

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**DEPURACION DE CODIGO**

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

En cualquier programa, tenemos tres tipos de errores:

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

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

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

Texto

El contenido generado por IA puede ser incorrecto.

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

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

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

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

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

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

El contenido generado por IA puede ser incorrecto.

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

Dos palabras clave:

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

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

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

Texto

El contenido generado por IA puede ser incorrecto.

**print("Operadores relacionales")**

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

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

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

**mayor = 0**

**menor = 0**

**intermedio = 0**

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

**mayor = num1**

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

**mayor = num2**

**else:**

**mayor = num3**

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

**menor = num1**

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

**menor = num2**

**else:**

**menor = num3**

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

**intermedio = suma - mayor - menor**

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

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

**BUCLES**

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

Tenemos dos tipos de bucles:

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

Sintaxis:

while (condicion == true):

#instrucciones

condicion = false

print(“Fin de programa”)

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

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

Sintaxis:

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

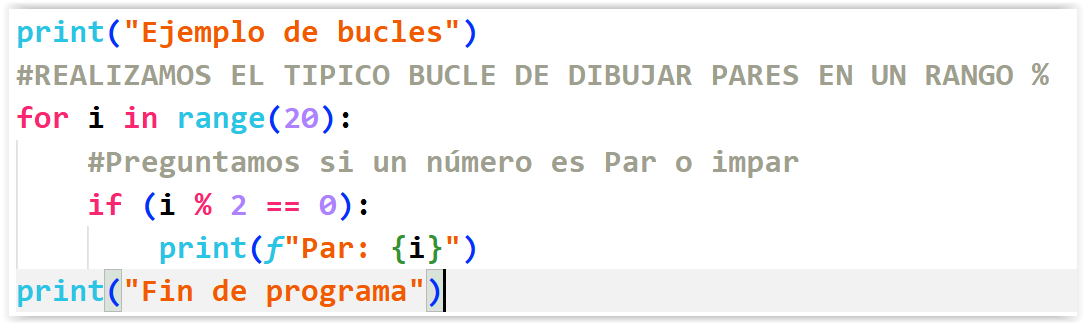
#instrucciones

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

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

#instrucciones

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



Vamos a crear un bucle para comprobar la Conjetura de Collatz

La conjetura de Collatz indica lo siguiente:

Todo número entero positivo será siempre 1 siguiendo estas instrucciones:

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

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

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**CLASE STRING PYTHON**

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

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

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

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

Un String es un conjunto de char/letras.

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

Un String diferencia mayúsculas de minúsculas en sus caracteres.

texto = “Lenguaje Python”

**len(texto)** 🡪 15

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

Todo conjunto comienza en CERO

texto[0] 🡪 L

texto[1] 🡪 e

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

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

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

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

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

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

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

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

texto = “primer python”

Tenemos dos posibilidades:

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

texto[8:] 🡪 ython

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

Texto[0:2] 🡪 pri

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

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

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

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

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

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

**# • Que el email contenga una @**

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

**# • @ ni al inicio ni al final**

**# • Solamente una @ en el mail**

**# • Un punto después de la @**

**# • Dominio de 2 a 3 caracteres**

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

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

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

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

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

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

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

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

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

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

**else:**

**#posicion del ultimo punto**

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

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

**#del ultimo punto**

**#paco@gmail.com**

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

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

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

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

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

**print("Email correcto")**

**else:**

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

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

Vamos a realizar un ejemplo para recorrer un String numérico.

Pediremos al usuario un texto de solo números.

Mostraremos la suma de dichos números.

Recorreremos carácter a carácter el texto y lo iremos sumando.

1234 🡪 10

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

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

El contenido generado por IA puede ser incorrecto.

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

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

**suma = 0**

**#1234**

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

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

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

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

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

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

**suma = suma + numero**

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

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

**METODOS/FUNCIONES DENTRO DEL LENGUAJE PYTHON**

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

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

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

La instrucción que hemos utilizado ha sido:

Primero el objeto:

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

Aplicamos el método al objeto email de la clase String

**email.startsWith(“@”)**

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

**len(**“texto”**)**

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

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

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

Todo funciona mediante llamadas y declaraciones de los métodos.

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

**PROGRAMA PRINCIPAL MAIN.PY**

print(“Mi programa principal”)

print(“Introduzca email”)

email = input()

**validarEmail()**

print(“Fin de programa”)

#METODO PARA VALIDAR UN EMAIL

**validarEmail()** {

print(“Validando email”)

#NUESTRO CODIGO DE VALIDACION

}

Mi programa principal

Introduzca un email

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

**Validando email**

Fin de programa

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

**PROGRAMA PRINCIPAL MAIN.PY**

print(“Mi programa principal”)

print(“Introduzca email”)

email = input()

**métodos.validarEmail()**

print(“Fin de programa”)

**METODOS.PY**

#METODO PARA VALIDAR UN EMAIL

**validarEmail()** {

print(“Validando email”)

#NUESTRO CODIGO DE VALIDACION

}

Mi programa principal

Introduzca un email

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

**Validando email**

Fin de programa

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

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

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

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

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

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

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

El contenido generado por IA puede ser incorrecto.

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

**#AL METODO**

***def* primerMetodo():**

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

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

***def* segundoMetodo():**

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

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

**#CODIGO PRINCIPAL DE NUESTRO PROGRAMA MAIN**

**print("Ejemplo de métodos")**

**#LLAMADA A LOS METODOS**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

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

## PARAMETROS EN LOS METODOS

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

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

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

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

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

Ejemplo:

def metodoParametros(**nombre**):

print(“Su nombre es “ + **nombre**)

Petición al método

metodoParametros(14)

metodoParametros(“Alumno”)

Método con múltiples parámetros:

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

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

print(“Su edad es “, edad)

Llamada al método

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

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

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

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

**#AL METODO**

***def* primerMetodo():**

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

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

***def* segundoMetodo():**

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

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

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

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

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

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

**#CODIGO PRINCIPAL DE NUESTRO PROGRAMA MAIN**

**print("Ejemplo de métodos")**

**saludar("Paco")**

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

**#saludar("Paco")**

**#LLAMADA A LOS METODOS**

**primerMetodo()**

**segundoMetodo()**

**primerMetodo()**

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