Programa educativo

Pytime

Capítulo profesional de la Sociedad de Computación de Ecuador (IEEE)

Facultad de Ingeniería en Electricidad y Computación (ESPOL)

Autores: Adriana Collaguazo Jaramillo , Axell Concha, Debbie Donoso, Angelo Sánchez, Josué Tomalá, William Ayala, Natalia Mawyin, Nathaly Indacochea, Steven Santillán, Christopher Vaccaro, Steven Nazareno, Carlos Cedeño, Ronny Benitez

Con la colaboración de:

espol Facultad de Ingeniería en Electricidad y Computación

Índice

1 | Fundamentos de programación

1.1 | Conceptos básicos

Python es un lenguaje de programación interpretado, de alto nivel y con semántica dinámica, esto nos indica que al momento de crear una variable en memoria, automáticamente asignará el tipo de variable a ese objeto [1]. Entre las características más destacables de Python se mencionan las siguientes:

- Es orientado a objetos: La programación orientada a objetos (POO) es notablemente fácil de aprender y aplicar con Python.
- Es libre: Es software de código abierto, no existen restricciones para copiarlo, integrarlo en sus sistemas o enviarlo con sus productos.
- Es portable: Python está escrito en ANSI C portátil, por eso se compila y ejecuta en prácticamente todas las plataformas principales como Windows, Linux, MacOS, entre otras.
- Es poderoso: Su conjunto de herramientas lo sitúa entre lenguajes de secuencias de comandos tradicionales (como Tcl, Scheme y Perl) y lenguajes de sistemas (como C, C++ y Java).
- Es combinable: Python puede fácilmente integrarse a componentes escritos en otros lenguajes de programación.
- Es fácil de usar: La combinación de respuesta rápida y simplicidad del lenguaje de Python hace que la programación sea más divertida que el trabajo.
- Es fácil de aprender: De hecho, puedes esperar estar codificando programas significativos en Python en cuestión de días (y quizás en sólo horas, si ya eres un programador experimentado).

En Python hay una gran colección de funcionalidades preconstruidas y portátiles, conocidas como librerías estándares, las cuales se pueden invocar con “import”. Estas librerías admiten una serie de tareas de programación a nivel de aplicación, desde la comparación de patrones de texto hasta la creación de scripts de red. Entre las librerías que usaremos se mencionan las siguientes:

- **time:** Este módulo ofrece varias funciones relacionadas con el tiempo.
- **pin:** Controla los pines de E/S, también conocido como General-Purpose Input/Output (GPIO). Los objetos pin se asocian comúnmente con un pin físico que puede controlar un voltaje de salida y leer voltajes de entrada.
- **dht:** El módulo dht proporciona funciones relacionadas con la lectura del sensor de temperatura y humedad de la serie dht.
- **ujson:** UltraJSON es un codificador y decodificador JSON escrito en C puro con enlaces para Python 3.7+.

Además, escribir en el lenguaje de Python es casi como escribir en pseudocódigo en inglés. Por ejemplo:

Pseudocódigo

```
Si e no está en [12,34,25,17,24]
    imprimir "No está en la lista"
```

Python 3

```
if e not in [12,34,25,17,24]:
    print("No está en la lista")
```

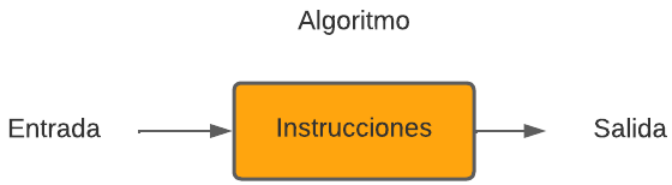

Figura 1.1: Algoritmo

1.2 | Intérprete y compilador

1.2.1 | Intérprete

Un intérprete es un tipo de programa que ejecuta otros programas. Cuando escribes un programa Python, el intérprete de Python lee tu programa y ejecuta las instrucciones que contiene. En efecto, el intérprete es una capa de lógica de software entre tu código y el hardware de tu máquina. Cuando el paquete de Python se instala en su máquina, genera una serie de componentes -mínimamente, un intérprete y una librería de apoyo-. Dependiendo de cómo lo utilices, el intérprete de Python puede adoptar la forma de un programa ejecutable o de un conjunto de librerías vinculadas a otro programa [1].

1.2.2 | Compilador

Un compilador traduce un programa a instrucciones específicas, es decir traduce código fuente de alto nivel a lenguaje de bajo nivel como lenguaje de máquina. Compilan y enlazan programas completos. La principal diferencia entre el intérprete y el compilador radica en el rendimiento. Una vez que se ejecuta el programa, los servicios del compilador ya no son necesarios, mientras que el intérprete continúa utilizando los recursos informáticos. En Python, el compilador está siempre presente en tiempo de ejecución y forma parte del sistema que ejecuta los programas.

1.3 | Algoritmos y sus representaciones

1.3.1 | Algoritmo

Un algoritmo es una sucesión ordenada de instrucciones que deben ejecutarse para resolver un problema en un tiempo finito. Su estructura se basa en la entrada de datos, estos son transformados produciendo una serie de acciones que se convertirán en la salida del algoritmo, como se muestra en la Figura 1.1.

Para desarrollar algoritmos se debe usar una metodología que permita resolver problemas de forma organizada. Primero definir el objetivo, luego identificar los componentes o variables y finalmente escribir las instrucciones que permitan llegar al objetivo propuesto. Se deben realizar pruebas para validar que el resultado sea el correcto.

1.3.2 | Ejemplo de algoritmo

Desarrollar un algoritmo que permita comprar entradas para la película Avengers desde la aplicación de Supercines. Para realizar la compra tenemos la información de una tarjeta de débito.

Para desarrollar el algoritmo primero identificamos nuestra entrada y nuestra salida esperada.

- **Entrada:** Información de la tarjeta de débito.
- **Salida esperada:** Entradas para la película.

Algoritmo

1. Ingresar a la aplicación de supercines
2. Seleccionar la película Avengers
3. Seleccionar el lugar
4. Seleccionar la hora de la película
5. Ingresar los datos de la tarjeta de débito para realizar el pago
6. Finalizar la compra

1.3.3 | Construcción de algoritmos

Para poder construir algoritmos computacionales se pueden usar diferentes notaciones. En este caso aprenderemos una notación simple y clara para ser usada en problemas básicos. Esta notación es útil cuando se está aprendiendo a diseñar algoritmos, una vez que se tenga mejor dominio del desarrollo, puede ser omitida.

Para esto se debe identificar los componentes que la comprende, cada componente incluye una instrucción simple o un conjunto de instrucciones. Para la representación gráficas representaremos cada componente como un bloque como se muestra en la Figura 1.2.

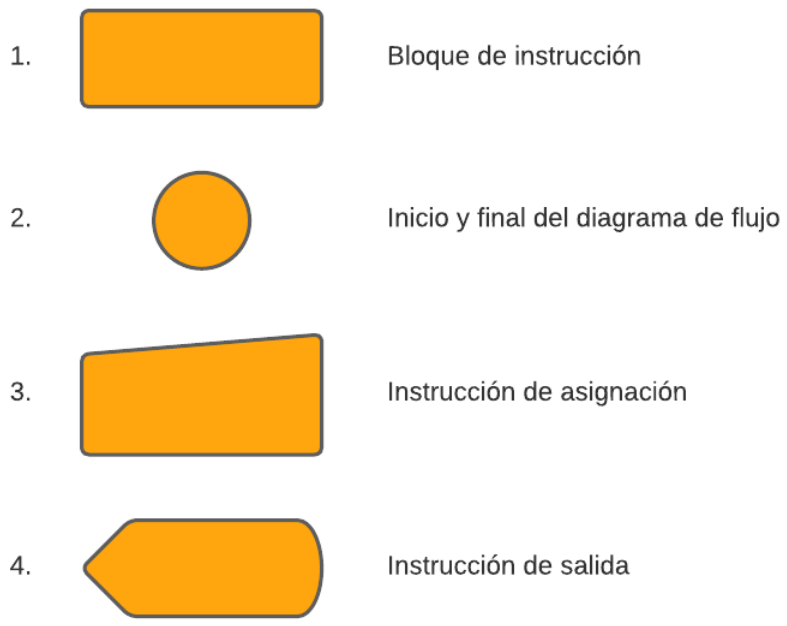

Figura 1.2: Bloques usados para elaborar diagramas de flujo

Un algoritmo no solo cuenta con una instrucción, sino una serie organizada de instrucciones que son ejecutadas secuencialmente, por lo que se muestra esta organización a través de líneas de flujo que unen cada uno de los bloques del componente. Además, encontramos los siguientes símbolos en los diagramas de flujo.

1. Bloque de instrucción: En este bloque se representa una acción o línea de código, usualmente se realizan operaciones con los operadores presentados más adelante.
2. Inicio y final del diagrama de flujo: Este bloque representa el inicio o fin del algoritmo.

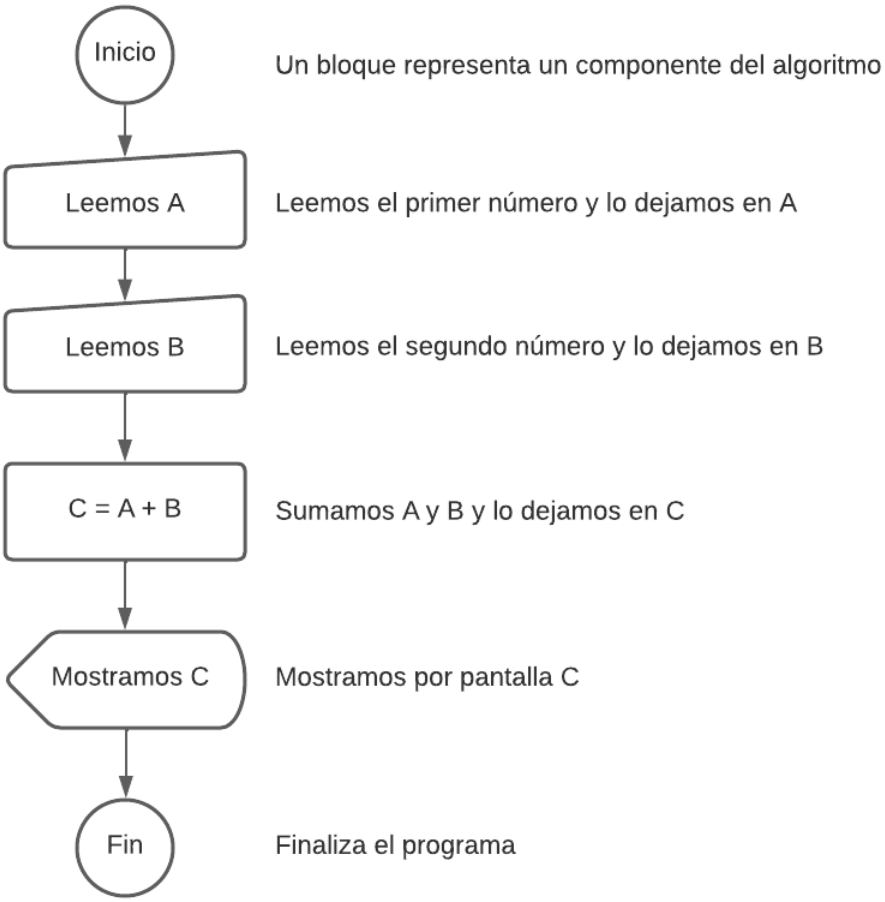

Figura 1.3: Ejemplo de diagrama de flujo

3. Instrucción de asignación: Este bloque permite representar la entrada por teclado, o asignación de un valor a una variable.
4. Instrucción de salida: Este bloque representa la salida esperada del algoritmo. También se puede usar para presentar una salida por pantalla.
5. Líneas de flujo: Estas permiten conectar los diferentes bloques para mostrar la secuencia que debe seguir el algoritmo.

Un ejemplo de un diagrama se muestra en la Figura 1.3.

1.4 | Ejercicios

Realizar los diagramas de flujo para representar los algoritmos de los siguientes procesos:

- Ingrese el nombre del usuario y se guarde en la variable *nombre*. Luego ingrese el apellido del usuario y se guarde en la variable *apellido*. Por último unir estas dos variables en una nueva variable llamada *nombre_completo* y mostrarlo por pantalla.
- El programa pide el costo de 3 productos A, B, y C. Se debe realizar la suma de los 3 productos y mostrar el resultado por pantalla.
- El programa pide el ingreso de la temperatura en Fahrenheit y retorna la temperatura convertida en Celsius.

2 | Tipos de datos

2.1 | Datos primitivos

Los datos primitivos son los que cubren los formatos básicos de los elementos que se utilizan para formar un programa. Estos son los caracteres numéricos y las cadenas de caracteres.

Los tipos de datos primitivos son los siguientes:

1. **int (enteros):** Son números sin punto decimal. Ejemplo: 4, 89, 230, -30
2. **float (números de punto flotante):** Números con punto decimal.
   Ejemplo: 20.6, 0.36, -52.9
3. **string (cadena de caracteres):** Hace referencia a un conjunto de caracteres, estas pueden ser una letra, una palabra, una oración, un párrafo, etc.; que son establecidos entre comillas simples o dobles.
   Ejemplo: “Hola”, “Welcome to python”
4. **bool (verdadero o Falso):** Este tipo de dato acepta valores por verdad = True o falso = False

2.2 | Variables

Las variables son contenedores de información que puede ir cambiando en el transcurso del tiempo de ejecución de un programa. Una variable está formada por un nombre que esté relacionado con el programa a desarrollar. Un valor, éste será el contenido de la variable. El nombre de la variable puede estar conformado por letras, números o el caracter guión bajo.

Por ejemplo, humedad, HUMEDAD, humedad_jardin.

Para python humedad y HUMEDAD son variables diferentes, ya que este lenguaje reconoce las mayúsculas y minúsculas como diferentes.

Reglas

- No debe iniciar con un número.
- El nombre no debe ser una **palabra reservada** de python. Las palabras reservadas tienen un significado especial para python por lo tanto no se debe usar como nombre de una variable.
  Ejemplos de palabras reservadas:

print	input	False	True	None	as	asset	break	continue
def	del	elif	else	except	finally	for	from	global
if	try	while	with	yield	not	in	and	input

Buenas prácticas

- Empezar el nombre de una variable con minúscula
- Elegir un nombre que represente el dato a guardar
  Ejemplos:

- temperatura
- humedad

- Nombres que contengan más de una palabra se debe capitalizar o separar con subguión cada palabra esta práctica se la conoce como **camelCase** y **snake_case** respectivamente.
  Ejemplos:
  - temperaturaCelsius
  - temperatura_celsius

Para crear una variable simplemente se debe establecer el nombre de la variable y asignando valor mediante el signo igual, por ejemplo:

Python 3

```
temperatura_celsius = 28
sensor = "DHT11"
```

Para presentar el contenido de una variable se usa la función print(), por ejemplo:

Python 3

```
print(temperatura_celsius)
```

Salida

```
28
```

2.3 | Operadores Aritméticos

En python existen operadores aritméticos que nos permiten calcular un valor de dos operandos. Los operadores aritméticos disponibles en python son: suma, resta, multiplicación, división, división entera, módulo y potenciación.

Operador	Operación	Ejemplo	Resultado
+	Suma	2 + 4	6
-	Resta	8 - 5	3
*	Multiplicación	6 * 2	12
/	División	9 / 2	4.5
//	División Entera	9 // 2	4
%	Módulo	9 % 2	1
**	Potenciación	2 ** 3	8

Ejemplo Suma

```
temperatura_celsius = 28
temperatura_celsius = temperatura_celsius + 1
print(temperatura_celsius)
```

Salida

29

Prioridad de Operadores

En la sección anterior se muestran ejemplos de operaciones simples con un operador, en python es posible realizar una operación más compuesta, pero hay que tener en claro la prioridad de los operadores, el orden de operación es el siguiente, siendo el paréntesis de mayor prioridad, suma y resta el de menor:

- Paréntesis ()
- Exponente **
- Multiplicación *, División /, División Entera //, Módulo %
- Suma +, Resta -

Conversión de Fahrenheit a Grados Celsius

Por ejemplo, supongamos que se posee un sensor de temperatura que lee en unidades Fahrenheit si deseamos convertir a grados Celsius podríamos hacer lo siguiente:

$$C = \frac{5(F-32)}{9}$$

```
temperaturaF = 82.4
temperaturaC = (5*(temperaturaF-32))/9
print(temperaturaC)
```

Salida

28

Ejercicio

```
a = 3.0
b = 4.0
c = (a ** 2 + b ** 2) ** 0.5
print(c)

#¿Cuál será el resultado?
```

2.4 | Operadores lógicos

Estos símbolos se utilizan para construir expresiones lógicas, nos permiten trabajar con valores de tipo booleano. Un valor booleano o bool es un tipo que solo puede tomar valores True o False. Por lo tanto, estos operadores nos permiten realizar diferentes operaciones con estos tipos y su resultado será otro booleano.

Matemáticas	Símbolo Python
Conjunción	and
Disyunción	or
Negación	not

Las tablas de verdad son las siguientes:

1. **Tabla de not**

A	not A
True	False
False	True

Ejemplo

```
print (not True)
print (not False)
print (not not not not True)
```

Salida

```
False
True
True
```

2. **Tabla de and**

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

Ejemplo 1

```
print (True and True)
print (True and False)
print (False and True)
print (False and False)
```

Salida

```
True
False
False
False
```

Ejemplo 2

```
x = 0
y = 10
print (x<5 and y>2)
```

Salida

```
True
```

Ejemplo 3

```
x = 0
y = 10
print (x and y)
```

Salida

```
0
```

3. **Tabla de or**

A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

Ejemplo 1

```
print (True or True)
print (True or False)
print (False or True)
print (False or False)
```

Salida

```
True
True
True
False
```

Ejemplo 2

```
x = 0
y = 10
print (x or y)
```

Salida

```
10
```

4. **Ejemplo general**

Ejemplo general

```
print (0 and not 1 or 1 and not 0 or 1 and 0)
```

Salida

```
True
```

2.5 | Conversión de datos

En Python, la conversión de tipos es un proceso en el que convertimos un literal de un tipo a otro. Las funciones incorporadas int(), float() y str() se utilizarán para el typecasting.

- **int()**
  Puede tomar un literal float o string como argumento y devuelve un valor de tipo class 'int'.
- **float()**
  Puede tomar como argumento un literal de int o de cadena y devuelve un valor de la class 'float'.
- **str()**
  Puede tomar un literal de float o int como argumento y devuelve un valor de la class 'str'.

1. **Conversión de tipo Entero (int) a Flotante (float)**
   El método de Python float() le permitirá convertir los enteros en flotantes.

Ejemplo 1

```
print (float(57))
```

Salida

```
57.0
```

También puede utilizar esto con una variable.

Ejemplo 2

```
f = 68
print (float(f))
```

Salida

```
68.0
```

Ejemplo 3

```
n = 100
f = float(n)
print (f)
```

Salida

```
100.0
```

2. **Conversión de tipo Entero (int) a Cadena (str)**

   Podemos convertir números en cadenas usando el método str().

Ejemplo 1

```
print ("El resultado es " + str(12))
```

Salida

```
El resultado es 12
```

También puede utilizar esto con una variable.

Ejemplo 2

```
numero = 50
print ("El resultado es " + str(numero))
```

Salida

```
El resultado es 50
```

Ejemplo 3

```
n = 100
s = "El resultado es " + str(n)
print (s)
```

Salida

```
El resultado es 100
```

3. **Conversión de tipo Flotante (float) a Entero (int)**

   Python también incluye una función integrada para convertir flotantes en enteros: int().

Ejemplo 1

```
print (int(390.8))
```

Salida

```
390
```

Ejemplo 2

```
b = 125.0
print (int(b))
```

Salida

```
125
```

Ejemplo 3

```
f = 100.05
n = int(f)
print (n)
```

Salida

```
100
```

Cuando se convierten flotantes en enteros con la función int(), Python corta el decimal y los números restantes de un flotante para crear un entero. Aunque posiblemente desee redondear 390.8 a 391, Python no lo hará con la función int().

4. **Conversión de tipo Flotante (float) a Cadena (str)**

Cuando colocamos un flotante en el método str(), se mostrará un valor de cadena del flotante.

Ejemplo 1

```
print ("El resultado es " + str(421.034))
```

Salida

```
El resultado es 421.034
```

Ejemplo 2

```
f = 5524.53
print ("El resultado es " + str(f))
```

Salida

```
El resultado es 5524.53
```

Ejemplo 3

```
f = 100.05
s = "El resultado es " + str(f)
print (s)
```

Salida

```
El resultado es 100.05
```

5. **Conversión de tipo Cadena (str) a Flotante (float)**

   Queremos convertir estas cadenas en flotantes con la función float().

Ejemplo 1

```
totalPuntos = "5524.53"
nuevosPuntos = "45.30"
resultado = float(totalPuntos) + float(nuevosPuntos)
print(resultado)
```

Salida

```
5569.83
```

Ejemplo 2

```
s = '132.65'
f = float(s)
print(f)
```

Salida

```
132.65
```

6. **Conversión de tipo Cadena (str) a Entero (int)**

   Si su cadena no tiene posiciones decimales, probablemente querrá convertirla en un entero utilizando el método int().

Ejemplo 1

```
temperaturaPasada = "34"
temperaturaActual = "28"
diferencia = int(temperaturaPasada) - int(temperaturaActual)
print(diferencia)
```

Salida

```
6
```

Ejemplo 2

```
s = "80"
i = int(s)
print(i)
```

Salida

```
80
```

Solo se realizará la conversión de datos si el contenido es compatible, sería ilógico intentar convertir **t45** a entero porque tiene una t, para algo así tendríamos que separar primero todo el texto.

2.6 | Manejo de entrada y salida

Para que un programa sea útil, generalmente necesita comunicarse con el mundo exterior obteniendo datos de entrada del usuario y mostrando los datos de resultados al usuario. Un desarrollador puede querer tomar la entrada del usuario en algún punto del programa. Para hacer esto, Python proporciona la función llamada **input()**.

Sintaxis

```
variable=input('solicitud')
```

Donde *'solicitud'*, es una cadena de caracteres **opcional**, que es mostrada al momento de pedir la entrada.

2.7 | Obtener entrada del usuario

La entrada del usuario es lo que una persona ingresa en el teclado, ya sea eso un caracter, una flecha presionada o tecla enter, u otra cosa. Para convertirla a cualquier otro tipo de datos, tenemos que hacerlo explícitamente.

Python 3

```
# Tomando entrada del usuario
nombre = input("Ingresa tu nombre: ")

# Salida
print("Hola, " + nombre)
print(type(nombre))
```

Salida

```
Ingresa tu nombre: Alex
Hola, Alex
<class 'str '>
```

2.8 | Usar entradas con enteros

Como se puede observar Python toma todas las entradas como una entrada de cadena de forma predeterminada. Para convertirlo a cualquier otro tipo de datos, tenemos que convertir la entrada explícitamente. Por ejemplo, para convertir la entrada a int o float tenemos que usar el método **int()** y **float()** respectivamente.

Python 3

```
# Tomando entrada de usuario como entero
edad = int(input( "Ingresa tu edad: " ))
nueva_edad = edad + 1
# Salida
print( "En tu siguiente cumpleaños tendrás" + nueva_edad + "años" )
```

Salida

```
Ingresa tu edad: 18
En tu siguiente cumpleaños tendrás 19 años
```

3 | Cadenas de caracteres

En términos de programación, generalmente llamamos *string* a una cadena. Cuando se piensa que una cadena es una colección de letras, el término tiene sentido. Todas las letras, números y símbolos de este folleto podrían ser una cadena. De hecho, su nombre podría ser una cadena, al igual que su número telefónico.

3.1 | Crear cadenas

En Python, creamos una cadena poniendo comillas alrededor del texto. Por ejemplo, podríamos etiquetar una cadena, así:

Python 3

```
monster = "¿Por qué los monstruos tienen tanto pelaje?"
Luego, para ver qué hay dentro de monster, podríamos escribir print(monster),
como esto:
print(monster)
```

Output

```
¿Por qué los monstruos tienen tanto pelaje?
```

También se pueden usar comillas simples para crear cadenas, así:

Python 3

```
monster = '¿Qué es morado y muy peludo?'
print(monster)
```

Salida

```
¿Qué es morado y muy peludo?
```

3.2 | Operaciones con cadenas

Python tiene una clase de cadena incorporada llamada **str** con muchas funciones útiles. Las cadenas pueden ser modificadas usando diversos operadores. A continuación se presentan algunos de los operadores de cadena más comunes.

Operador de concatenación "+" Dos cadenas se pueden concatenar o unir usando el operador + en python, como se explica en el siguiente ejemplo:

Python 3

```
string1 = "PyTime"
string2 = "IoT"
cadena = string1+string2
print(cadena)
```

Salida

```
PyTime IoT
```

Operador de repetición de cadenas "*"

La misma cadena se puede repetir en python n veces usando cadena*n, como se explica en el siguiente ejemplo:

Python 3

```
string1 = "PyTime"
print(string1*2)
print(string1*3)
print(string1*4)
print(string1*5)
```

Salida

```
PyTime PyTime
PyTime PyTime PyTime
PyTime PyTime PyTime PyTime
PyTime PyTime PyTime PyTime PyTime
```

Operador de división de cadenas "[]"

Se puede acceder a los caracteres de un índice específico de la cadena con el operador cadena[índice]. El índice se interpreta como un índice positivo a partir de 0 desde el lado izquierdo y un índice negativo a partir de -1 desde el lado derecho.

cadena	H	E	L	L	O
índice positivo	0	1	2	3	4
índice negativo	-5	-4	-3	-2	-1

- **string[a]** Devuelve un carácter de un índice positivo de la cadena del lado izquierdo como se muestra en el gráfico de índice anterior.
- **string[-a]** Devuelve un carácter de un índice negativo a de la cadena del lado derecho como se muestra en el gráfico de índice anterior.

- **string[a:b]** Devuelve caracteres del índice positivo a al índice positivo b de como se muestra en el gráfico de índice anterior.
- **string[a:-b]** Devuelve caracteres del índice positivo a al índice negativo b de la cadena como se muestra en el gráfico de índice anterior.
- **string[a:]** Devuelve caracteres desde el índice positivo a hasta el final de la cadena.
- **string[:b]** Devuelve los caracteres desde el inicio de la cadena hasta el índice positivo b.
- **string[-a:]** Devuelve caracteres desde el índice negativo a hasta el final de la cadena.
- **string[:-b]** Devuelve los caracteres desde el inicio de la cadena hasta el índice negativo b.
- **string[::-1]** Devuelve una cadena en orden inverso.

Python 3

```
string1 = "PyTime"
print(string1[1])
print(string1[-3])
print(string1[1:5])
print(string1[1:-3])
print(string1[2:])
print(string1[:5])
print(string1[:-2])
print(string1[-2:])
print(string1[::-1])
```

Salida

```
y
i
yTim
yT
Time
PyTim
PyTi
me
emiTyP
```

Operadores de comparación de cadenas “==” “!=”

El operador de comparación de cadenas en python se usa para comparar dos cadenas.

- El operador “==” devuelve Boolean True si dos cadenas son iguales y devuelve Boolean False si dos cadenas no son iguales.
- El operador “!=” devuelve Boolean True si dos cadenas no son iguales y devuelve Boolean False si dos cadenas son iguales.

Estos operadores se utilizan principalmente junto con la condición if para comparar dos cadenas en las que la decisión se debe tomar en función de la comparación de cadenas.

Python 3

```
string1 = "PyTime"
string2 = "PyTime, IoT"
string3 = "PyTime, IoT"
string4 = "IoT"
print(string1==string4)
print(string2==string3)
print(string1!=string4)
print(string2!=string3)
```

Salida

```
False
True
True
False
```

Operador de pertenencia “in” y “not in”

El operador de pertenencia se usa para buscar si el carácter específico es parte/miembro de una cadena de Python de entrada determinada.

- “a” en la cadena: devuelve True booleano si “a” está en la cadena y devuelve False si “a” no está en la cadena.
- “a” no está en la cadena: devuelve True booleano si “a” no está en la cadena y devuelve False si “a” está en la cadena.

Un operador de membresía también es útil para encontrar si una subcadena específica es parte de una cadena dada.

Python 3

```
string1 = "PyTimeIoT"
print("i" in string1)
print("I" in string1)
print("a" in string1)
print("a" not in string1)
print("pytime" in string1)
print("Pytime" in string1)
print("pytime" not in string1)
```

Salida

```
True
True
False
True
False
False
True
```

3.3 | Diferencia entre cadenas de caracteres y números

La entrada del usuario ingresa en Python como una cadena, lo que significa que cuando escribe el número 10 en su teclado, Python guarda el número 10 en una variable como una cadena, no como un número, pero ¿Cuál es la diferencia entre el número 10 y la cadena '10'? Ambos nos parecen iguales, con la única diferencia de que uno está rodeado de comillas. Pero para una computadora, los dos son muy diferentes.

Python 3

Por ejemplo, supongamos que comparamos el valor de la variable temperatura con un número:

```
comparar = temperatura == 10:
```

Luego establecemos la variable temperatura en el número 10:

```
temperatura = 10
comparar = temperatura == 10:
print("¿Son iguales?", comparar)
```

Salida

```
¿Son iguales? True
```

Como puede ver, el resultado de la expresión booleana es **True**

Python 3

A continuación, establecemos temperatura en la cadena '10' (entre comillas), así:

```
temperatura = '10'
comparar = temperatura == 10:
print("¿Son iguales?", comparar)
```

Salida

```
¿Son iguales? False
```

Aquí, el resultado es **False** porque Python no ve el número en comillas (una cadena) como un número. Afortunadamente, Python tiene funciones mágicas que pueden convertir cadenas en números y números en cadenas. Por ejemplo, puedes convertir la cadena '10' en un número con **int()**:

Python 3

```
temperatura = '10'
temperatura_convertida = int(temperatura)
comparar = temperatura_convertida == 10:
print("¿Son iguales?", comparar)
```

Salida

```
¿Son iguales? True
```

3.4 | Operador de formato de cadena " %"

El operador de formato de cadena se utiliza para formatear una cadena según el requisito. Para insertar otro tipo de variable junto con la cadena, se usa el operador " %"junto a la cadena de python. " %" tiene el prefijo de otro carácter que indica el tipo de valor que queremos insertar junto con la cadena de Python. Consulte la tabla a continuación para conocer algunos de los diferentes especificadores de formato de cadena comúnmente utilizados:

OPERADOR	DESCRIPCIÓN
%d	Entero decimal con signo
%s	Cadena
%f	Número real de punto flotante

Python 3

```
sensor = "DTH11"
temperatura = 19.0
humedad = 74

ejemplo1 = "El sensor usado es: %s" % (sensor)
print(ejemplo1)

ejemplo2 = "Temperatura: %f °C, Humedad: %d" % (temperatura,humedad)
print(ejemplo2)
```

Salida

```
El sensor usado es: DTH11
Temperatura: 19.0 °C, Humedad: 74
```

Ejercicio

Realice un programa que usando las variables temperatura y humedad del ejemplo anterior, muestre la siguiente salida:

Salida

```
{ 'temperatura': 19.0, 'humedad':74 }
```

4 | Sentencia condicional

4.1 | Sentencia if

If es una estructura de control que nos permite ejecutar un bloque de código cuando se cumpla una expresión booleana. Podemos identificar dos partes principales en la estructura de control if (ver Figura 4.1):

- **Expresión booleana:** es la condición que se debe cumplir para que se ejecute bloque de código.
- **Bloque de código:** son las acciones a realizar en caso de cumplirse la expresión booleana.

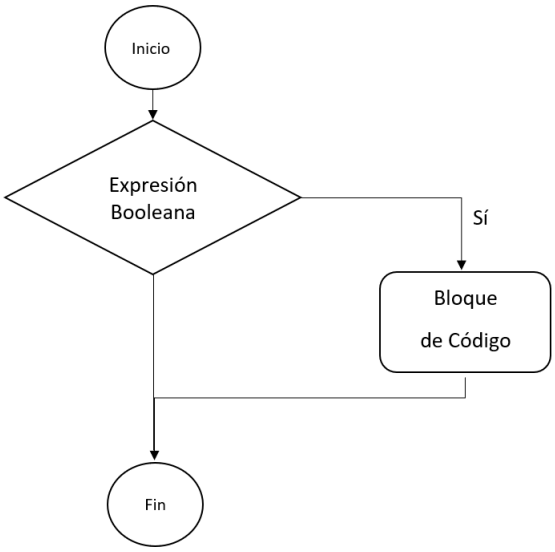

Figura 4.1: Diagrama de flujo de la estructura de control if

Por ejemplo, si el porcentaje de humedad del jardín es menor a 40 % se debe encender el sistema de riego, en un diagrama de flujo se observa de la siguiente manera

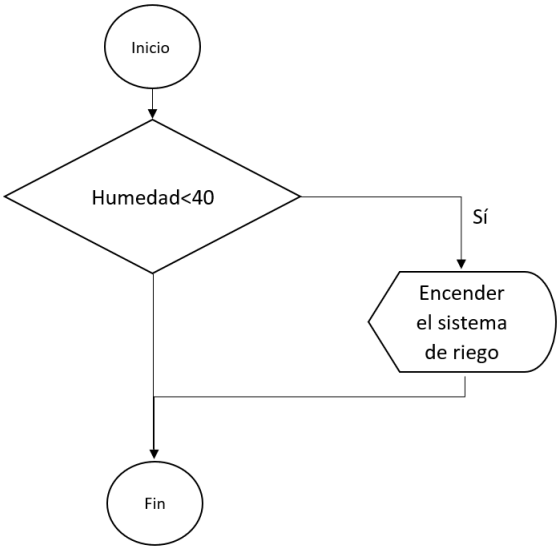

Figura 4.2: Ejemplo de diagrama de flujo de sentencia condicional

Python 3

```
humedad = 34
print("Inicio del programa")
if humedad < 40:
    print("Encender el sistema de riego")
print("Fin del programa")
```

Salida

```
Inicio del programa
Encender el sistema de riego
Fin del programa
```

Python 3

```
humedad = 50
print("Inicio del programa")
if humedad < 40:
    print("Encender el sistema de riego")
print("Fin del programa")
```

Salida

```
Inicio del programa
Fin del programa
```

4.2 | Sentencia if... else...

En la sección anterior observamos como se ejecutaba un bloque de código cuando la expresión en la sentencia **if** daba como resultado **True**, caso contrario se continuaba con el programa. Si se desea ejecutar código en el caso que no se cumpla la sentencia **if**, podemos incluir la sentencia **else**, el diagrama sería el siguiente

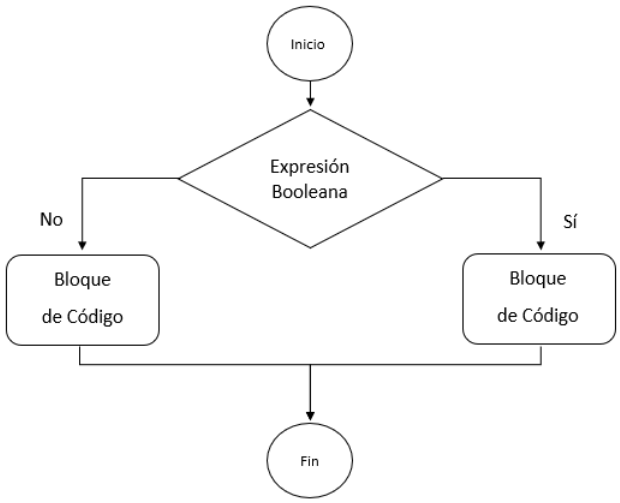

Figura 4.3: Diagrama de flujo de la estructura de control if... else...

Python 3

```
humedad = 50
print("Inicio del programa")
if humedad < 40:
    print("Encender el sistema de riego")
else:
    print("El jardín está en excelentes condiciones")
print("Fin del programa")
```

Salida

```
Inicio del programa
El jardín está en excelentes condiciones
Fin del programa
```

5 | Bucles o lazos

Una vez aprendido los conceptos básicos de programación, los tipos y operadores de datos, presentamos una *estructura de control* para ejecutar acciones repetitivas según una condición dada.

Pero, ¿Qué es una *estructura de control*?. Es un bloque o sección de código que nos permite agrupar instrucciones de manera controlada. Esto es fundamental al desarrollar sistemas puesto que hay aplicaciones donde tenemos que realizar acciones repetitivas y no siempre vamos a saber cuántas veces esto va a ocurrir. A continuación lo explicamos.

5.1 | Bucle While

Supongamos que estamos en una habitación y deseamos mostrar la temperatura del ambiente mediante una aplicación escrita con Python y la ayuda de sensores. En principio queremos mostrar la temperatura **dos** veces. A continuación se muestra su representación en diagrama de flujo y código.

Python 3

Para el siguiente código, vamos a asumir que el comando *sensor.leer()* nos devuelve la temperatura del ambiente.

```
temperatura = sensor.leer()
print("La temperatura del ambiente es: " + temperatura + "ºC")
temperatura = sensor.leer()
print("La temperatura del ambiente es: " + temperatura  + "ºC")
```

Salida

```
La temperatura del ambiente es 24.1 ºC
La temperatura del ambiente es 24.5 ºC
```

Si ahora queremos mostrar **cinco** veces el valor de temperatura deberíamos repetir el fragmento de código donde leemos el valor del sensor e imprimimos por pantalla las veces deseadas. Pero, ¿Y si queremos mostrarlo

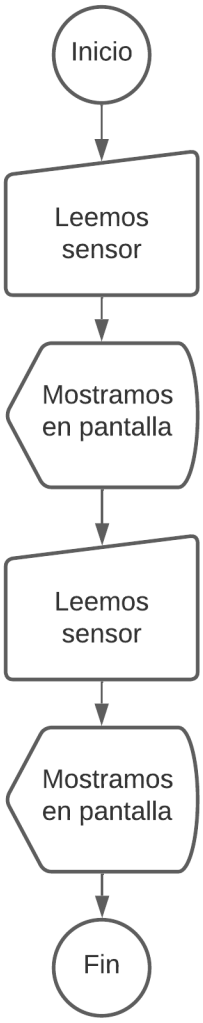

Figura 5.1: Ejemplo de algoritmo para mostrar secuencialmente dos lecturas de sensor

cien veces? esto de repetir el código se volvería extenso. Incluso ¿Qué pasa si queremos repetirlo por siempre o bajo una cierta condición? Aquí entra en juego el **Bucle While**.

5.2 | Creación de Bucle While

Podemos identificar dos partes principales en el Bucle While (ver Figura 5.2):

- **Expresión de control:** es la condición que se debe cumplir para que se ejecute el código.
- **Bloque de código:** son las acciones a realizar dentro del Bucle.

¡Ten cuidado! Un mal uso del Bucle While puede dar lugar a ciclos infinitos y la ejecución del programa puede no ser la deseada. Tratamos de siempre tener una condición de escape que permita terminar el Bucle While. Esta condición suele estar relacionada con el cambio de una variable.

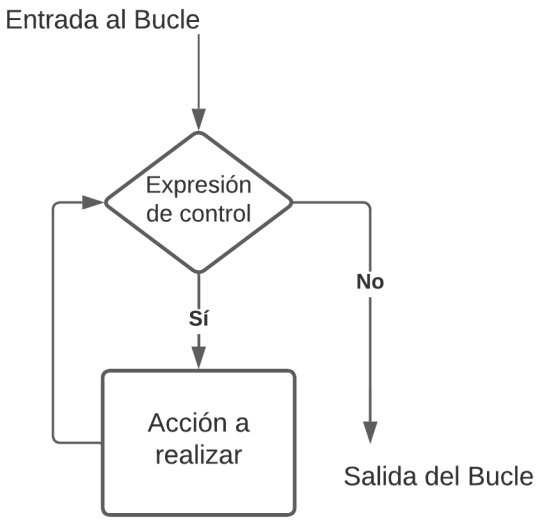

Figura 5.2: Equivalente en diagrama de flujo del Bucle While

Por ejemplo, siguiendo el problema planteado en la subsección anterior, ahora queremos mostrar la temperatura **solo** cuando sea menor a 25ºC. Una vez que la temperatura sea mayor o igual a 25ºC el programa terminará. A continuación te presentamos el código.

Python 3

Para el siguiente código, vamos a asumir que el comando *sensor.leer()* nos devuelve la temperatura del ambiente.

```
temperatura = sensor.leer()
while temperatura < 25:
    print("La temperatura del ambiente es: " + temperatura + "ºC")
    temperatura = sensor.leer()
print("La temperatura ha alcanzado los 25ºC")
```

Salida

```
# La temperatura oscila entre 24 y 24.9 ºC

La temperatura del ambiente es 24.1 ºC
La temperatura del ambiente es 24.5 ºC
La temperatura del ambiente es 24.7 ºC

# La temperatura pasa los 24.9 ºC

La temperatura ha alcanzado los 25ºC
```

Existen casos donde si deseamos que el programa se quede en un ciclo infinito, por ejemplo, si quisiéramos mostrar **siempre** la temperatura independientemente del cambio. Para ello podemos hacer uso del dato booleano *True* como se muestra a continuación.

Python 3

Para el siguiente código, vamos a asumir que el comando *sensor.leer()* nos devuelve la temperatura del ambiente.

```
temperatura = sensor.leer()
while True:
    print("La temperatura del ambiente es: " + temperatura + "ºC")
    temperatura = sensor.leer()
print("Esta línea nunca se va a imprimir")
```

Salida

```
# La temperatura toma cualquier valor
```

La temperatura del ambiente es 24.1 ºC
La temperatura del ambiente es 24.5 ºC
La temperatura del ambiente es 24.7 ºC
.
.

```
# Continuará hasta que se interrumpa la ejecución del programa
```

6 | Práctica de laboratorio de IoT con librería micro-python

Objetivo de aprendizaje: Leer datos en tiempo real usando una placa de desarrollo de hardware ESP32 con librería micro-python enviando datos a la nube de Ubidots por el protocolo de comunicación MQTT.

Recursos de hardware: 1 ESP32, 1 sensor DHT11, 1 protoboard, 10 jumpers tipo macho-macho, 2 LEDs, 2 resistencias de 330 ohmios.

Recursos de software: Plataforma de IoT de Ubidots, Visual Studio Code, micropython, librerías.

Duración: 120 minutos.

Introducción

Micropython: Es un pequeño pero eficiente interprete del Lenguaje de Programación Python, optimizado para funcionar en microcontroladores y ambientes restringidos. Con MicroPython, se pueden realizar muchas tareas, como controlar entradas/salidas de un microcontrolador como hacer parpadear un LED, obtener lecturas de señales analógicas y digitales, controlar servomotores, pantallas OLED, además de realizar comunicaciones I2C, SPI. Algunas características de microPython:

- Lenguaje de programación Python.
- Dispone de multitud de librerías para la ejecución de tareas.
- Es extensible, pueden extender de python a funciones más bajo nivel como C o C++.

ESP32: El módulo ESP32 es un microcontrolador de bajo costo y de bajo consumo de energía con tecnología WiFi y Bluetooth de modo dual integrado. Fue fabricado y desarrollado por la empresa Espressif Systems, una empresa con sede en Shanghai que cuenta con un historial grande de ofrecer productos de calidad en el área de los microcontroladores.

Message Queuing Telemetry Transport (MQTT): Es un protocolo de comunicación Machine to Machine (M2M) de tipo cola de mensajes. Está basado en el modelo base para la comunicación TCP/IP. Cada conexión se mantiene abierta y se reutiliza en cada comunicación. MQTT funciona por medio de un servicio de mensajería push que utiliza un patrón publicador/suscriptor (pub-sub). En este tipo de infraestructuras los clientes se conectan a un servidor central denominado bróker. Los clientes inician una conexión TCP/IP con el bróker, el cual mantiene un registro de los clientes conectados. Esta conexión se mantiene abierta hasta que el cliente la finaliza. Para filtrar los mensajes que son enviados a cada cliente los mensajes se disponen en tópicos organizados jerárquicamente. Un cliente puede publicar un mensaje en un determinado tópico, otros clientes pueden suscribirse a este tópico y el bróker le hará llegar los mensajes suscritos.

Ubidots: Es una plataforma de Internet of Things (IoT) que permite enviar datos de sensores a la nube, configurar tableros y alertas, conectarse con otras plataformas, usar herramientas de analíticas y arrojar mapas de datos en tiempo real.

Escenario

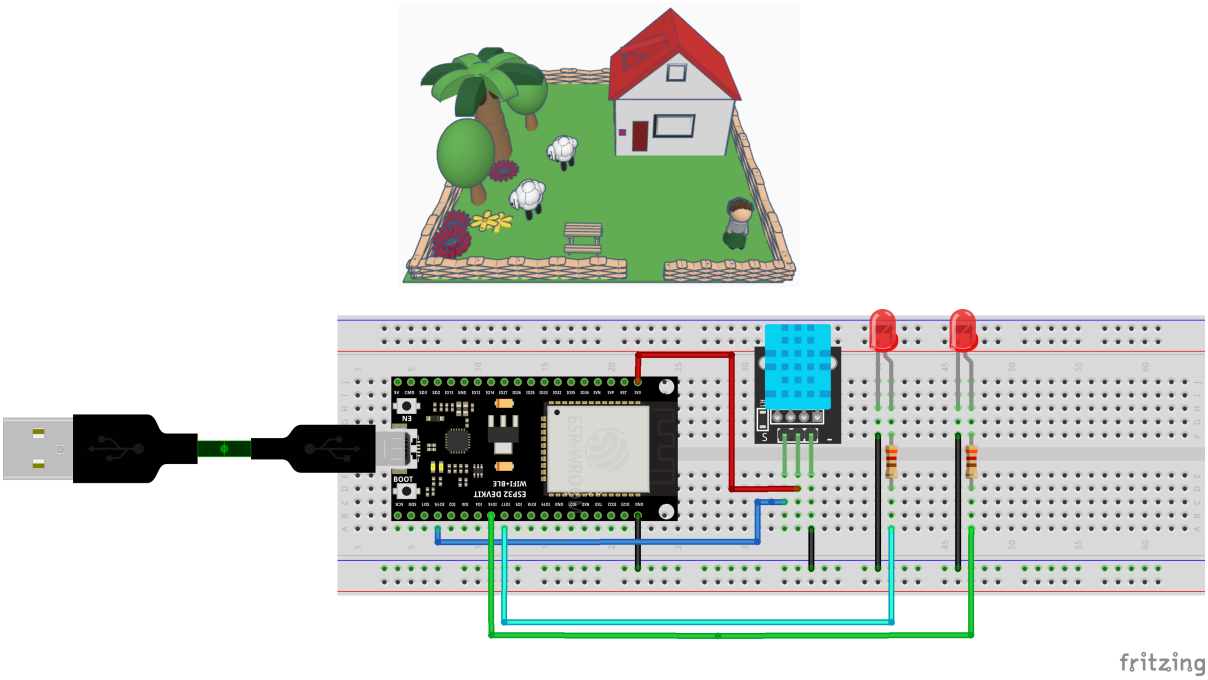

Figura 6.1: Diagrama de circuito de la práctica de IoT

Procedimiento

Se utilizará código fuente y herramientas necesarias como microprocesador, sensor, cable USB, jumpers, resistencias para una ejecución y visualización de datos en tiempo real sobre un escenario real como se muestra en la Figura 6.1. De esta manera podemos medir los valores del sensor DHT11, para presentarlos en una plataforma IoT.

Paso 0 (Informativo): Instalación del firmware de micropython.

- a) Abrir una ventana del terminal de su PC para instalar la herramienta esptool.

Comando

```
pip install esptool
```

Nota: Si el comando anterior no funciona, intente instalar esptool.py con los comandos que se muestran a continuación.

Comandos

```
pip3 install esptool
python -m pip install esptool
pip2 install esptool
```

- b) Instalar setuptools ya que no se encuentra habilitado en todos los sistemas de forma predeterminada.

Comando

```
pip install setuptools
```

Ahora, para verificar que la instalación se ha realizado de forma correcta ejecutamos el siguiente comando:

Comando

```
python -m esptool
```

Si la instalación se realizó sin problemas, como se muestra en la Figura 6.2:

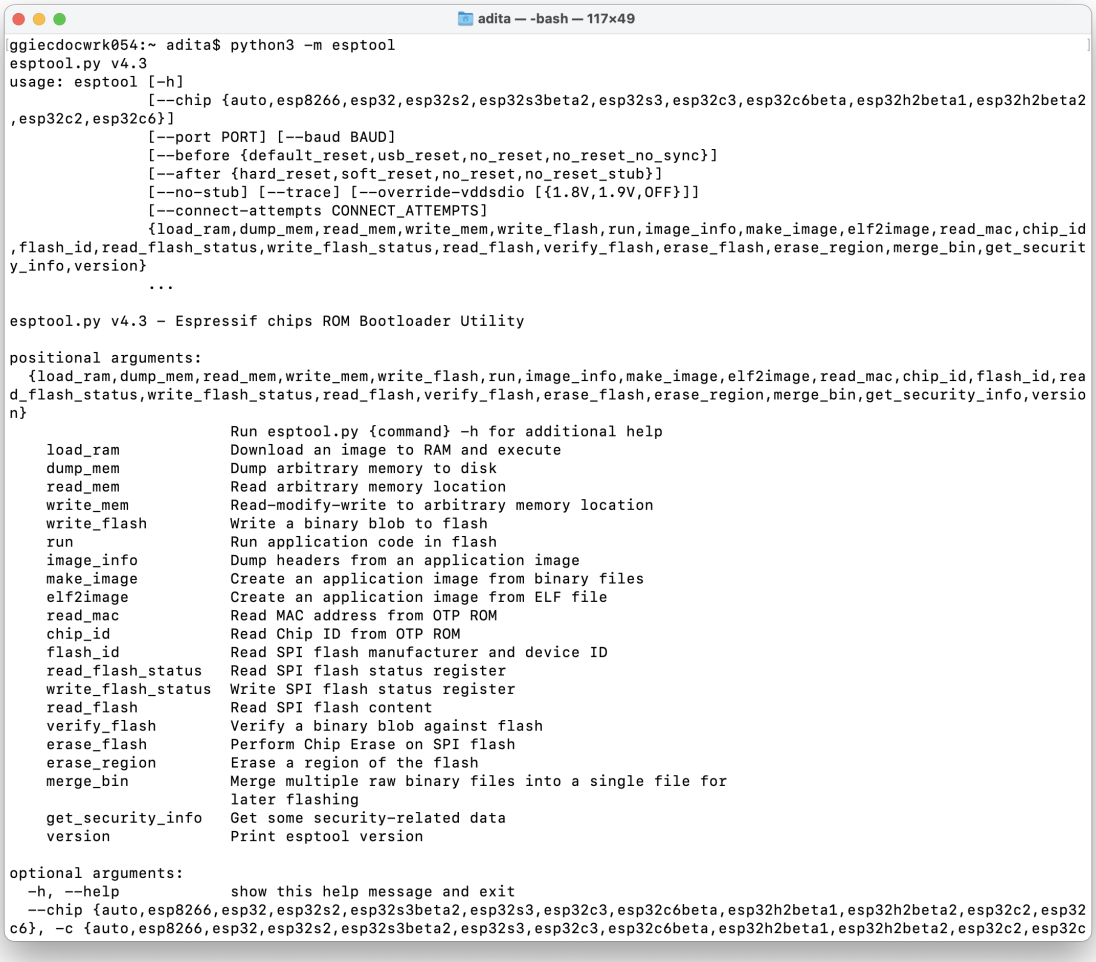

Figura 6.2: Instalación exitosa

c) Ingresar al enlace https://micropython.org/download/esp32/ de descarga del firmware de micropython para ESP32 y escogemos la última versión disponible como se muestra en la Figura 6.3.

d) Previo a la instalación del firmware se debe borrar la memoria flash de la ESP32. Para eso, mantenga presionado el botón **IO (BOOT)** que se muestra en un recuadro azul en la Figura 6.4.

Luego, ejecutar el siguiente comando:

Comando

```
python -m esptool --chip esp32 erase_flash
```

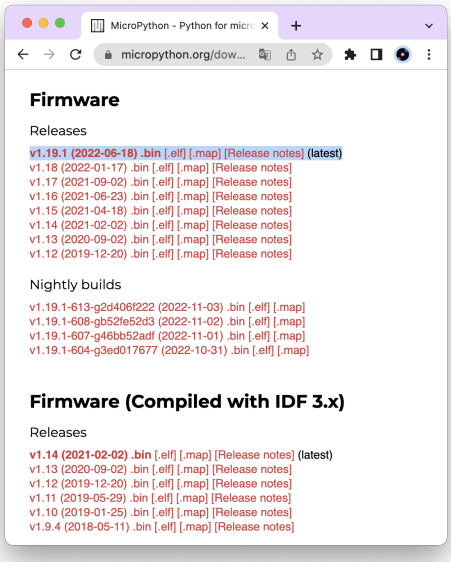

Figura 6.3: Firmware de micropython

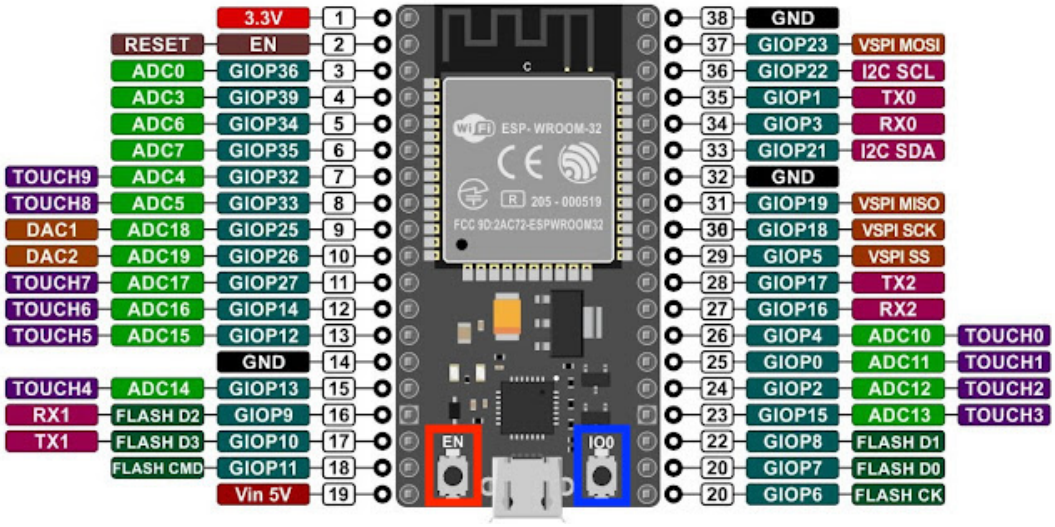

Figura 6.4: Botones de la ESP32

Cuando comience el proceso de “borrado” se puede soltar el botón de “BOOT”. Después de unos segundos, la memoria flash del ESP32 se borrará.

- e) Una vez que la memoria flash este borrada, se puede instalar el firmware de MicroPython usando el comando mostrado, donde debe reemplazar las variables <puerto_serie><esp32-X.bin>:

Comando

```
python -m esptool --chip esp32 --port <puerto_serie> write_flash -z 0x1000
<esp32-X.bin>
#<puerto_serie> será el puerto COM en el que el esp32 esté conectado en la PC
#<esp32-X.bin> será el nombre del .bin descargado en el literal c)
```

Nota: Mantener presionado el botón "BOOT" en el ESP32 antes de ejecutar el comando y esperar a que termine el proceso.

Paso 1: Creación del proyecto en Visual Studio Code (VS Code).

- a) Abrir la aplicación de escritorio de VS Code.
- b) En la pantalla principal de Visual Studio Code nos dirigimos a la extensión de Pymakr y damos clic sobre ella, luego escogemos "Create project", ver Figura 6.5.

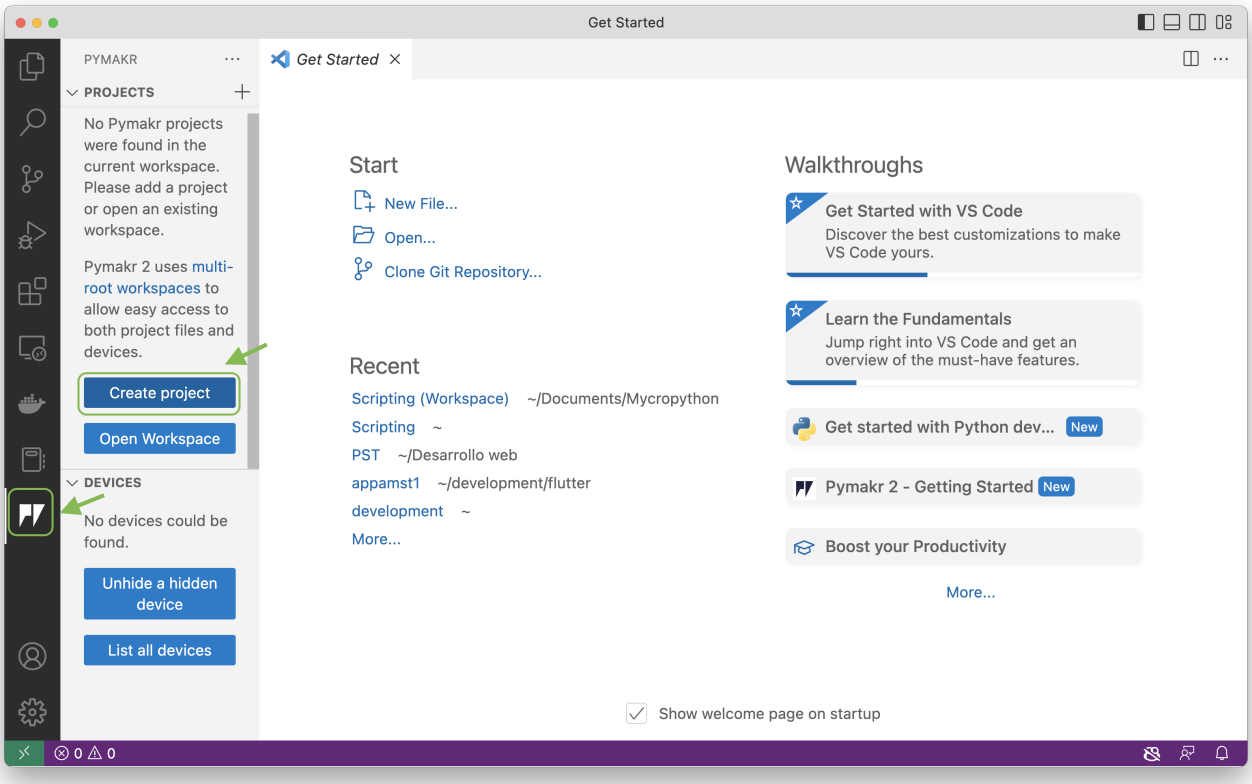

Figura 6.5: Extensión de pymark

- c) Se abrirá una ventana donde crearemos una carpeta en cualquier ruta (de preferencia en Escritorio) hacemos clic en "Use this folder" y finalmente damos un enter para confirmar el nombre del proyecto, como se muestra en la Figura 6.6.
- d) Luego escogemos la opción que dice "empty", como se muestra en la Figura 6.7.

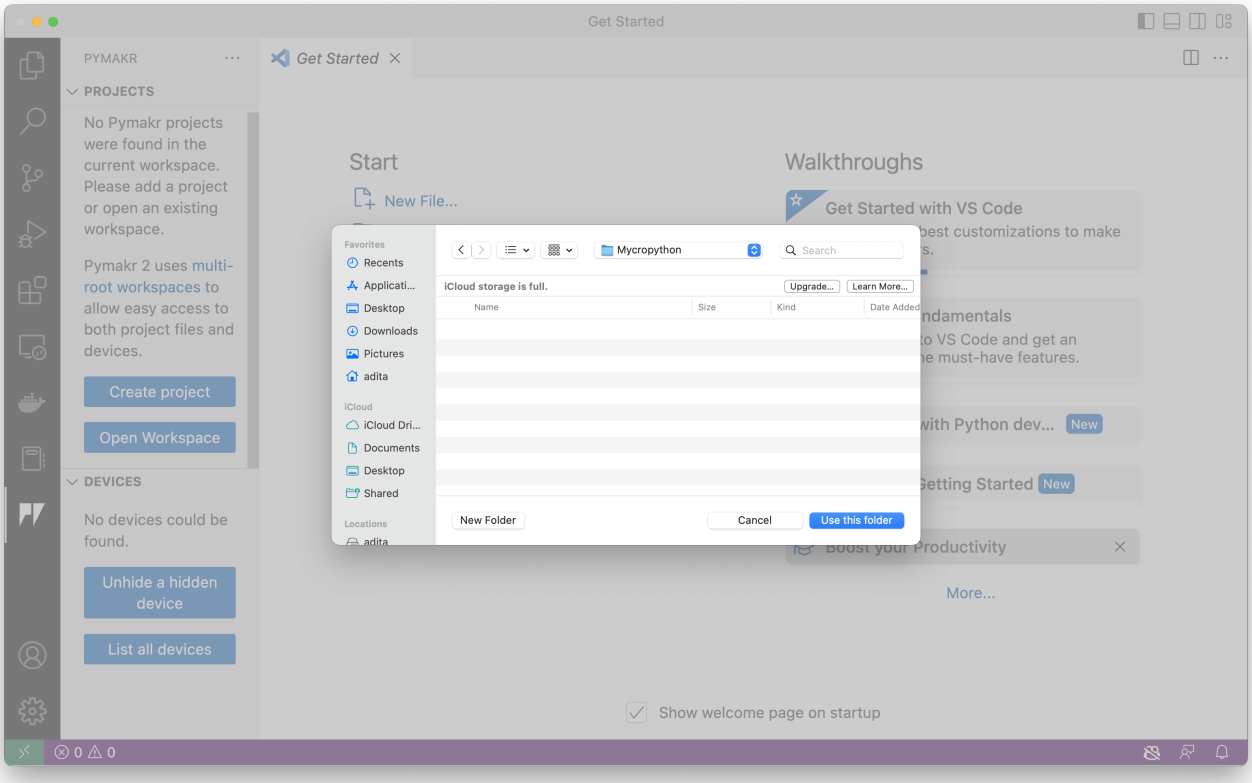

Figura 6.6: Ruta de la carpeta

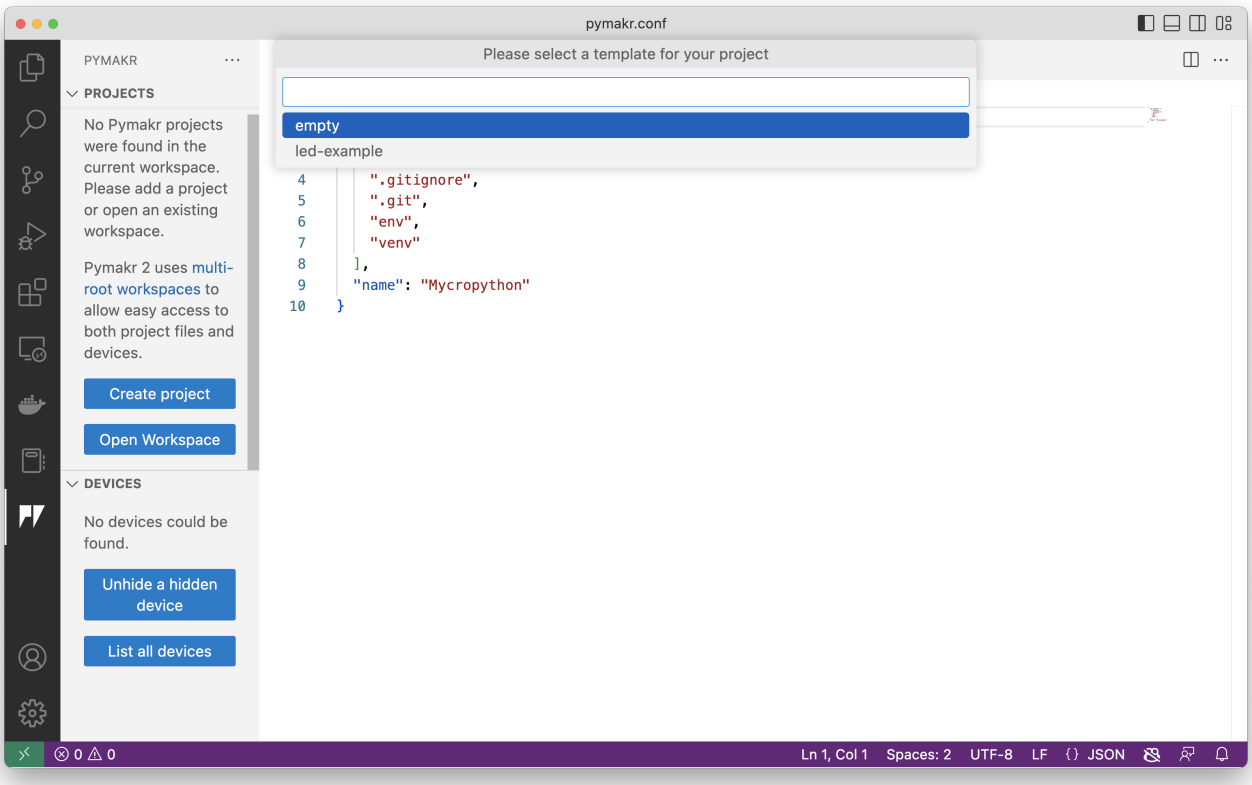

Figura 6.7: Proyecto vacío

- e) Al seleccionar la opción empty, se creará un nuevo proyecto. Ver Figura 6.8.

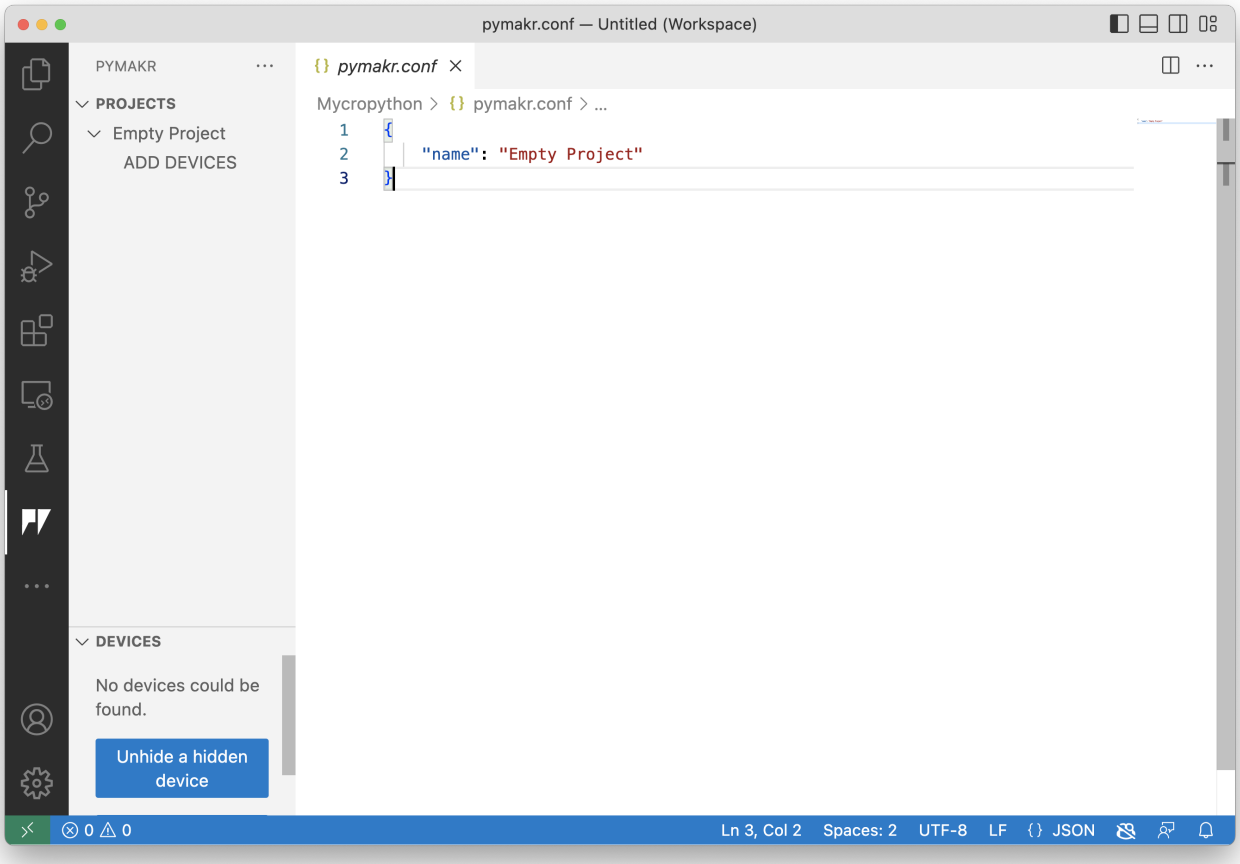

Figura 6.8: Nuevo proyecto en VS Code

Paso 2: Programación de la lectura y envío de datos.

- a) Seleccionar la opción de explorador de archivos, la cual mostrará 3 archivos: boot.py, main.py, y pymakr.conf. Ver Figura 6.9

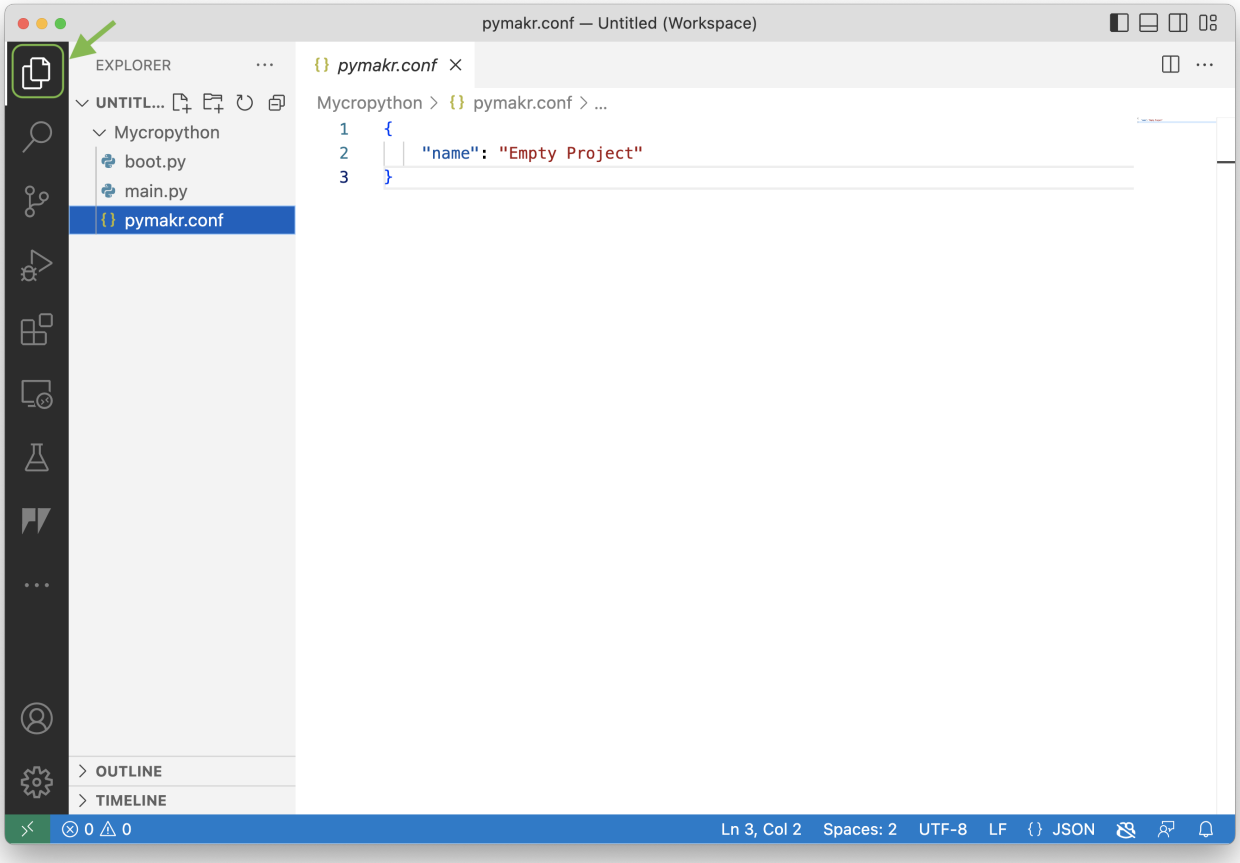

Figura 6.9: Archivos creados en el proyecto de VS Code

- b) Programar el sensor dando clic en el archivo main.py, como se muestra en la Figura 6.10.

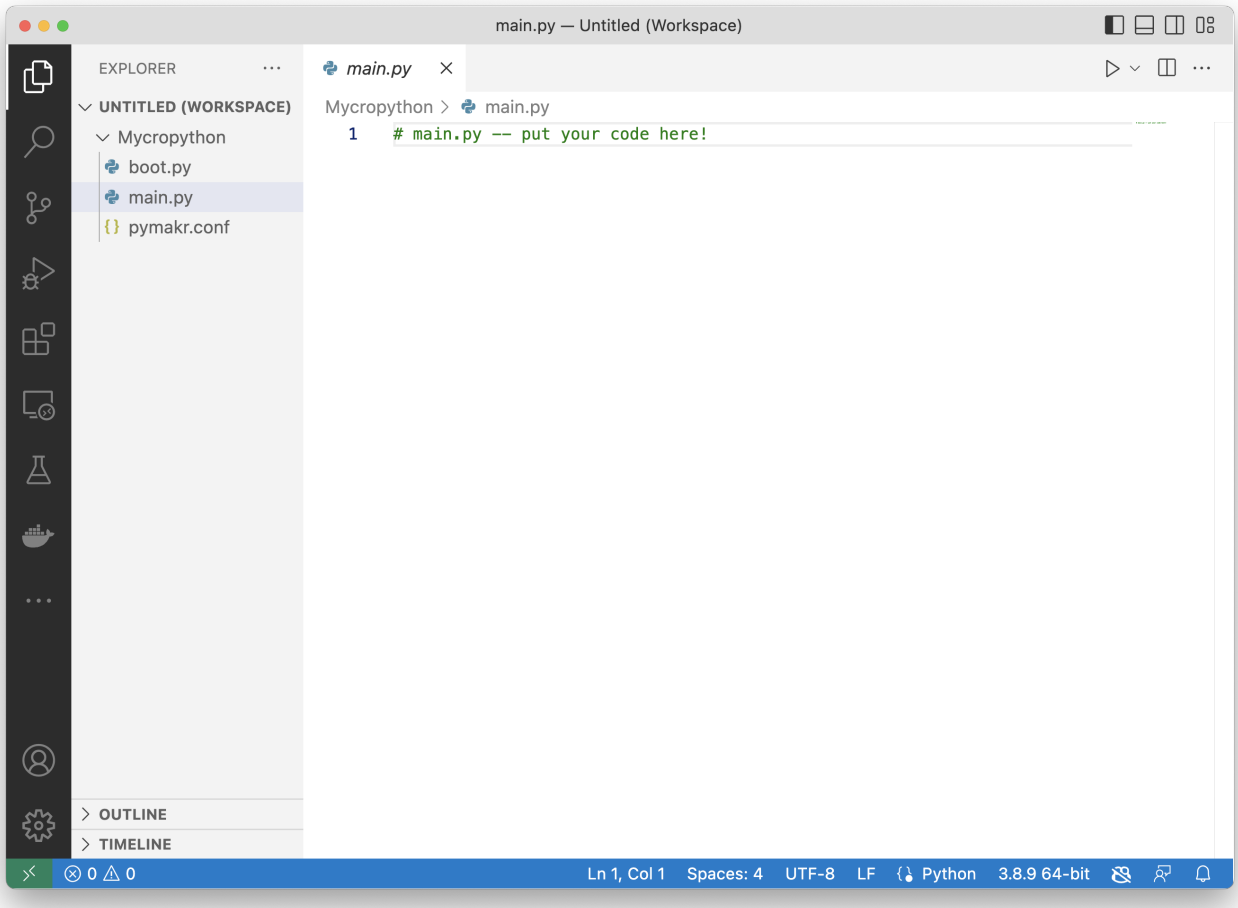

Figura 6.10: Archivo de Python main.py

- c) Importar librerías necesarias en el archivo de micropython. A continuación, se listan las librerías que se utilizarán para la implementación del presente taller:

Pyhton: main.py

```
import time
from machine import Pin
import dht
import ujson
```

- d) Definir el sensor a utilizar con el pin correspondiente de lectura en el ESP32.

Python: main.py

```
pin_led1 = 16
pin_led2 = 17
sensor = dht.DHT11(Pin(15))
led1 = Pin(pin_led1,Pin.OUT)
led2 = Pin(pin_led2,Pin.OUT)
```

- e) Bloque de código de realización de lectura de los datos.

Python: main.py

```
prev_weather = ""
while True:
  print("Obteniendo valores... ")
  sensor.measure()
# se declara la variable datos que tiene tem y hum
  datos = {
    "temperatura": sensor.temperature(),
    #declaramos la variable temp del sensor
    "humedad": sensor.humidity(),
  }
  message = ujson.dumps(datos)
  if message != prev_weather:
    print("Valores actualizados!")
    print(message)
    if (datos["temperatura"]<23):
      print("leds encendidos")
      led1(1)
      led2(1)
    else:
      print("led apagado")
      led1(0)
      led2(0)
    prev_weather = message
  else:

    print("Sin cambios" )
  time.sleep(2)
```

- f) Código completo de lectura de datos del sensor DHT11.

```
import time
from machine import Pin
import dht
import ujson
pin_led1 = 16
pin_led2 = 17
sensor = dht.DHT11(Pin(15))
led1 = Pin(pin_led1,Pin.OUT)
led2 = Pin(pin_led2,Pin.OUT)
#El número 15, el número 16 y 17 pueden cambiar de
#acuerdo al pin al que se conecte el sensor DHT11.
prev_weather = ""
#Variable en donde se guardarán los datos de
#temperatura y humedad.

while True:
  print("Obteniendo valores... ")
  sensor.measure()
  #Se declara la variable datos que tiene temp y hum
  datos = {
    #declaramos la variable temperatura, la cual guarda el valor
    #de temperatura leído
    "temperatura": sensor.temperature(),
    #declaramos la variable humedad, la cual guarda el valor
    #de humidity leído
    "humedad": sensor.humidity(),
  }
  message = ujson.dumps(datos)
# Validamos que el valor de temp y hum cambie y en caso de que no sea
#así se imprime "Sin cambios"
  if message != prev_weather:
    print("Valores actualizados!")
    print(message) #Se imprimen los valores de temp y hum
    if (datos["temperatura"]<23):
      print("led encendido")
      led1(1)
      led2(1)
    else:
      print("led apagado")
      led1(0)
      led2(0)
    prev_weather = message
  else:
    print("Sin cambios" )
  time.sleep(2)
```

En el siguiente paso, se configurarán los parámetros del servidor MQTT para obtener el token de la plataforma Ubidots. Para eso, debe contar con una cuenta en dicha plataforma.

- **Paso 3:** Creación de cuenta en la plataforma de IoT de Ubidots.

  a) En caso de poseer una cuenta debe dirigirse directamente al paso 4.

  b) Ingresar al enlace https://industrial.ubidots.com/accounts/signup_industrial/ para crear la cuenta en Ubidots.

c) Seleccionar el botón TAKE ME TO UBIDOTS STEM.

d) Se presentará el formulario de registro en el que debe colocar un nombre de usuario, correo electrónico, contraseña, seleccionar la opción "My IoT project for personal non-commercial use: y luego hacer clic en SIGN UP FOR FREE, como se muestra en la Figura 6.11.

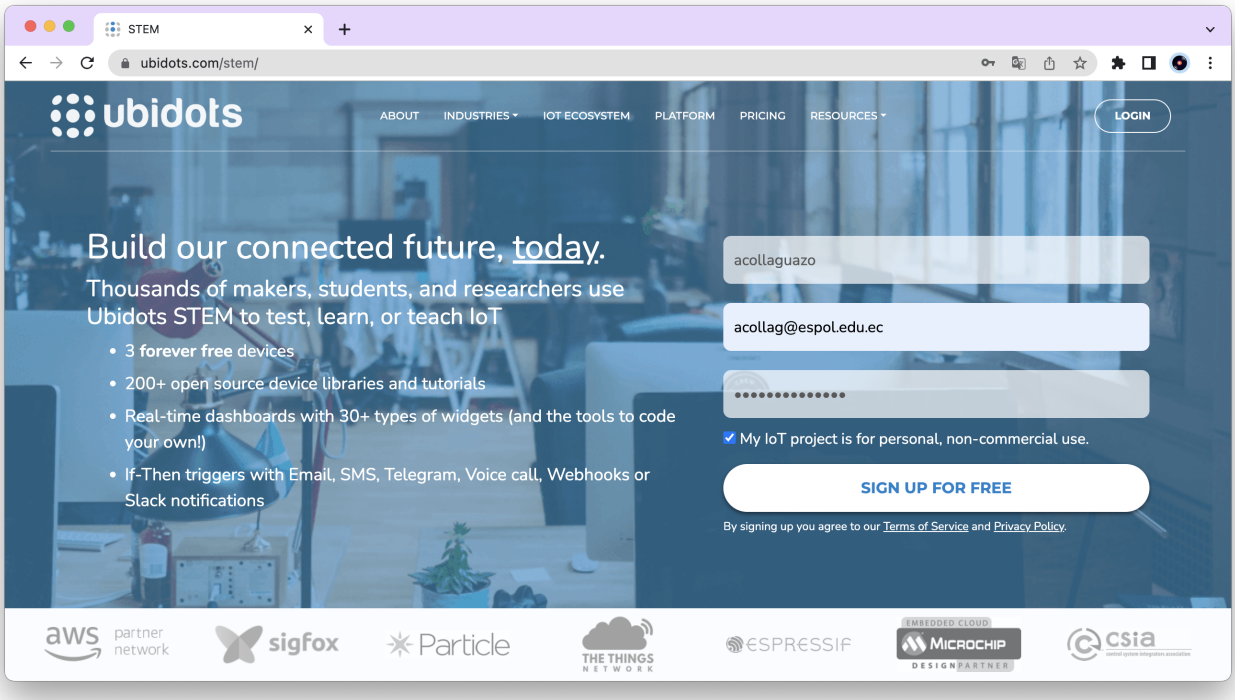

Figura 6.11: Plataforma de Ubidots

e) Luego de que la cuenta haya sido creada se presentará una ventana de bienvenida de Ubidots. Dar clic en la flecha que se encuentra en la esquina inferior derecha de la ventana. Ver Figura 6.12.

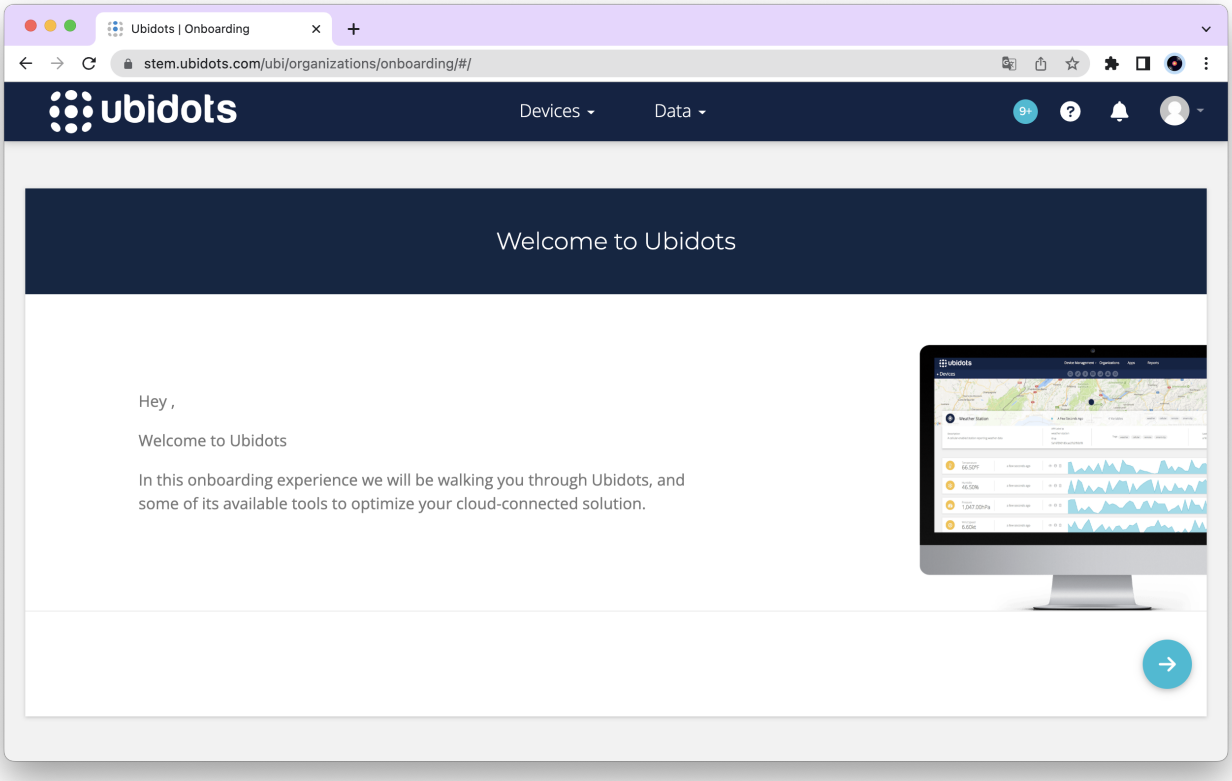

Figura 6.12: Pantalla de bienvenida de Ubidots

f) Continuar dando clic en la flecha que se encuentra en la esquina inferior derecha hasta que se presente la ventana de Thank You. Luego dar clic en el visto ubicado en la esquina inferior derecha. Ver Figura 6.13.

g) Se mostrará la pantalla de Demo Dashboard de Ubidots, como sigue en la Figura 6.14.

- **Paso 4:** Obtención del token de la plataforma Ubidots para la configuración del servidor MQTT.

  a) En el panel de control de Ubidots, seleccionamos icono del Usuario que se encuentra en la esquina superior derecha. Luego, se mostrará una lista de opciones seleccionamos **API Credentials**, copiamos el token y lo pegamos en un bloc de notas para luego usarlo en el código de micropython, como se muestra en la Figura 6.15.

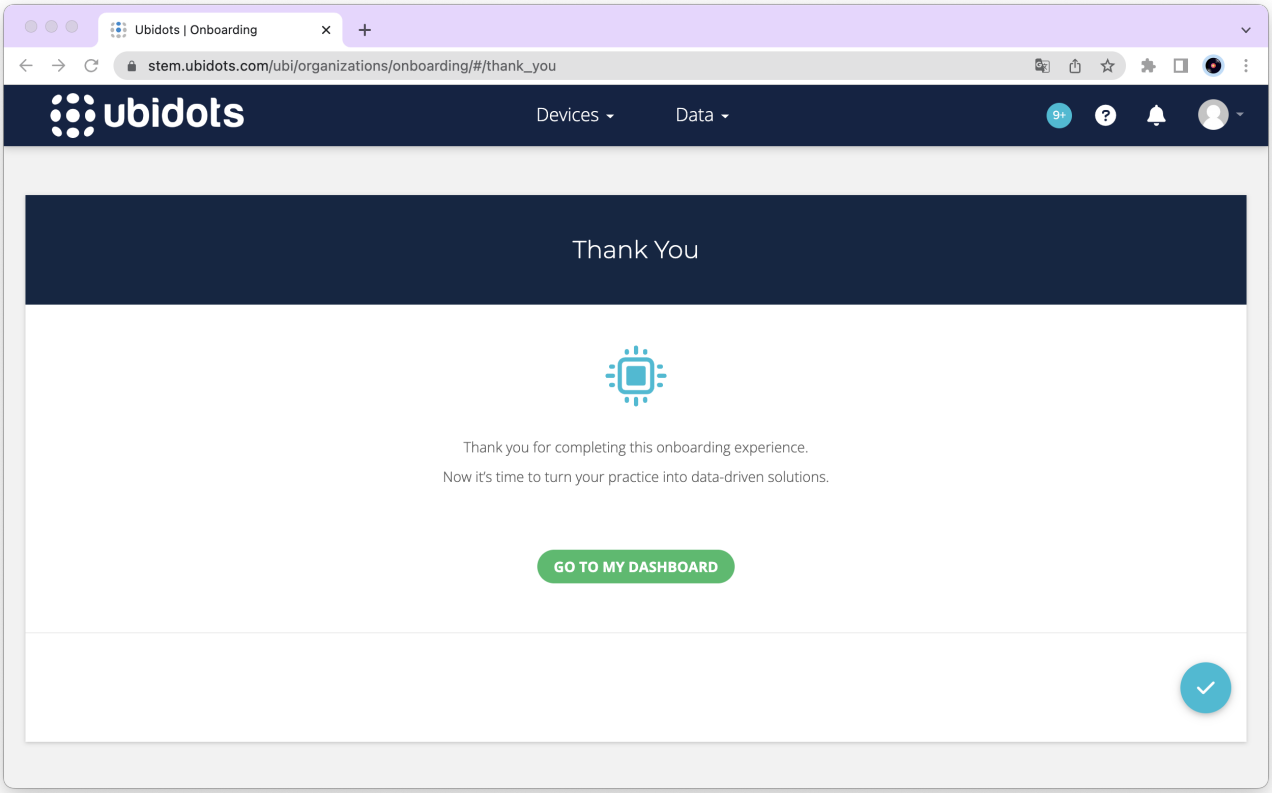

Figura 6.13: Finalización del tutorial de Ubidots

- **Paso 5:** Configuración de la conectividad mediante WiFi y del servidor MQTT.

  a) Importar librerías de conexión a la red WiFi y al servidor MQTT.

  Python: main.py

```
import network
from umqtt.simple import MQTTClient
```

  b) Definición de parámetros del servidor MQTT.

  Python: main.py

```
# MQTT Server Parameters
MQTT_CLIENT_ID = "micropython-weather-demo"
MQTT_BROKER    = "industrial.api.ubidots.com"
MQTT_USER      = "PEGAR-AQUÍ-EL-TOKEN-OBTENIDO"
MQTT_PASSWORD  = " "
MQTT_TOPIC     = "IEEE-PyTime-IOT" #copiar incluido comillas
```

  c) Realizamos la configuración de los parámetros de conexión a la red WiFi.

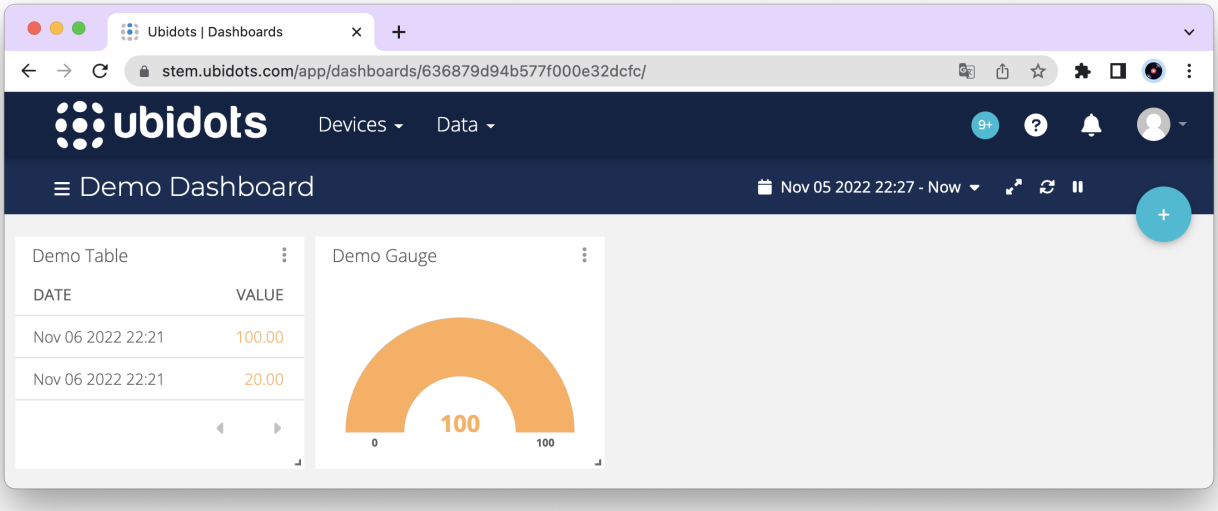

Figura 6.14: Panel de control de Ubidots

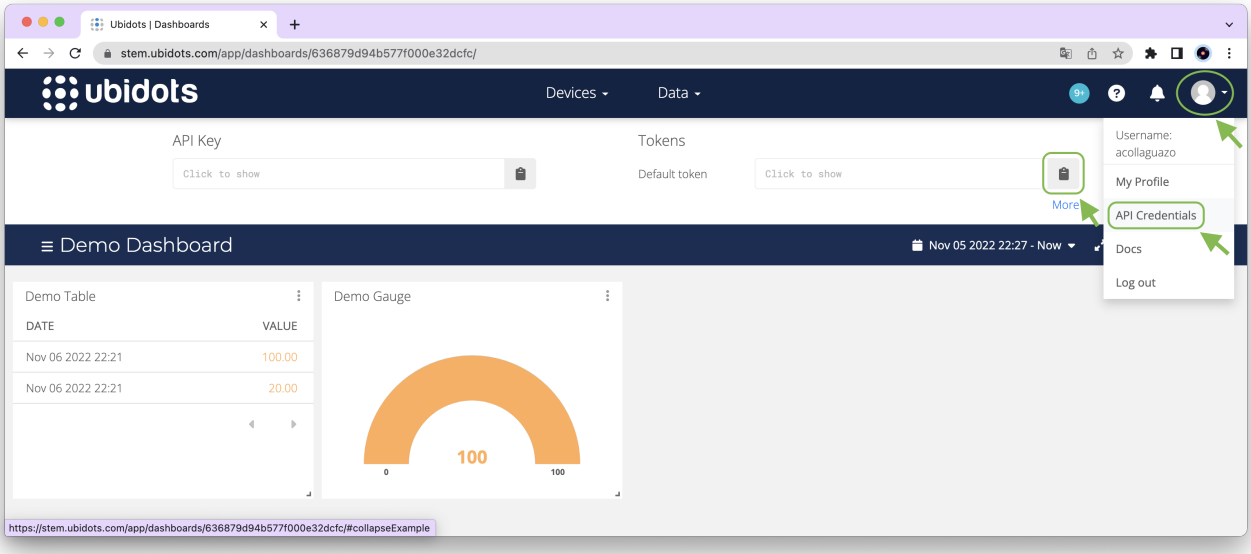

Figura 6.15: Token del API de Ubidots

Python: main.py

```
print("Connecting to WiFi", end="")
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
sta_if.connect('SSID-RED', 'PASSWORD-RED') #parametros de la red
while not sta_if.isconnected():
    print(".", end="")
    time.sleep(0.1)
print(" Connected!")
```

d) Ahora procedemos a realizar la conexión de servidor MQTT usando los parámetros establecidos en el literal b.

Python: main.py

```
print("Connecting to MQTT server... ", end="")
client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER, user=MQTT_USER,
password=MQTT_USER)
client.connect()

print("Connected!")
```

■ **Paso 6:** Utilización del protocolo MQTT para el envío de datos a Ubidots.

a) Modificación del código de lectura dado en el Paso 2.

Python: main.py

```
#Reutilizamos el código del paso 2
prev_weather = ""

while True:
  print("Measuring weather conditions... ", end="")
  sensor.measure()
  datos = {
    "temperatura": sensor.temperature(),
    "humedad": sensor.humidity(),
  }
  message = ujson.dumps(datos)
  # Validamos que el valor de temperatura y humedad cambie
  # y en caso de que no un mensaje.
  if message != prev_weather:
    print("Updated!")
    print("Reporting to MQTT topic {}: {}".format(MQTT_TOPIC, message))
    client.publish(b"/v1.6/devices/ESP32",message)
    if (datos["temperatura"]<23):
      print("led encendido")
      led1(1)
      led2(1)
    else:
      print("led apagado")
      led1(0)
      led2(0)
    prev_weather = message
  else:
    print("No change")
  time.sleep(1)
```

Nota: En el bloque condicional se resalta la implementación del método format. Este método se encarga de formatear la cadena que lo invoca, es decir, la cadena que llama a este método puede contener texto o campos de reemplazo delimitados por llaves {}. Si no se especifica nada dentro de las llaves se asigna cada elemento en el orden que se definen las variables dentro de format. Dado

a que no se especifica nada dentro de las llaves se asigna MQTT_TOPIC en las primeras llaves y message en las segundas.

b) Código completo de la práctica.

```
import time
from machine import Pin
import dht
import ujson
import network
from umqtt.simple import MQTTClient

# MQTT Server Parameters
MQTT_CLIENT_ID = "micropython-weather-demo"
MQTT_BROKER = "industrial.api.ubidots.com"
MQTT_USER = "PEGAR-AQUÍ-EL-TOKEN-OBTENIDO"
MQTT_PASSWORD = " "
MQTT_TOPIC = "IEEE-PyTime-IOT" #copiar incluido comillas

pin_led1 = 16
pin_led2 = 17
sensor = dht.DHT11(Pin(15))
led1 = Pin(pin_led1,Pin.OUT)
led2 = Pin(pin_led2,Pin.OUT)
#El número 15, 16 y 17 pueden cambiar de acuerdo al pin al
#que se conecte el sensor DHT11.

print("Connecting to WiFi", end="")
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
sta_if.connect('SSID-RED', 'PASSWORD-RED') #parametros de la red
while not sta_if.isconnected():
    print(".", end="")
    time.sleep(0.1)
    print("Connected!")

print("Connecting to MQTT server... ", end="")
client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER, user=MQTT_USER,
password=MQTT_USER)
client.connect()
print("Connected!")

#reutilizamos el código del paso 2
prev_weather = ""
while True:
  print("Measuring weather conditions... ", end="")
  sensor.measure()
  datos = {
   "temperatura": sensor.temperature(),
   "humedad": sensor.humidity(),
  }
  message = ujson.dumps(datos)
  # Validamos que el valor de temperatura y humedad cambie
  # y en caso de que no un mensaje.
  if message != prev_weather:
    print("Updated!")
    print("Reporting to MQTT topic {}: {}".format(MQTT_TOPIC, message))
    client.publish(b"/v1.6/devices/ESP32",message)
```

```
    if (datos["temperatura"]<23):
      print("led encendido")
      led1(1)
      led2(1)
    else:
      print("led apagado")
      led1(0)
      led2(0)
    prev_weather = message
  else:
    print("No change")
  time.sleep(1)
```

- **Paso 7:** Visualización de los datos de humedad y temperatura por consola y en la plataforma IoT de Ubidots.

  a) Subir el código fuente del archivo de python main.py a la ESP32 usando VS Code en el icono "Sync project to device", como se muestra en la 6.16.

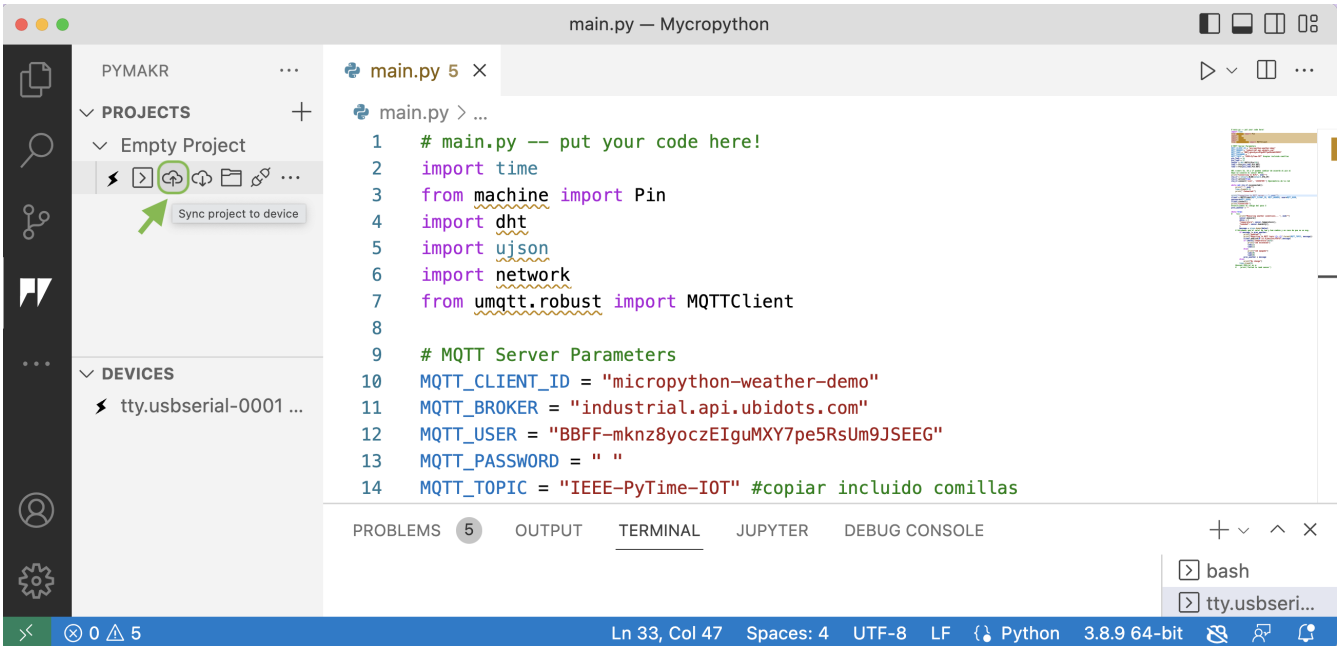

Figura 6.16: Subir código fuente en ESP32

  b) En la consola de VS Code presionar las teclas de Control + d para un soft reset. Otra opción es reiniciar la ESP32 usando el botón de **EN** que se muestra en un recuadro rojo en la figura 6.4.

  c) Se observará en la consola los datos de lectura del sensor DHT11 por el protocolo de comunicación MQTT, como se muestra en la Figura 6.17.

  d) Nos dirigimos a la plataforma de Ubidots en la sección de **"Device"**. Ver Figura 6.18.

```
>>>
MPY: soft reboot
Connecting to WiFiConnecting to MQTT server... Connected!
Measuring weather conditions... Updated!
Reporting to MQTT topic IEEE-PyTime-IOT: {"humedad": 52, "temperatura": 25}
led apagado
Measuring weather conditions... Updated!
Reporting to MQTT topic IEEE-PyTime-IOT: {"humedad": 51, "temperatura": 24}
led apagado
Measuring weather conditions... No change
Measuring weather conditions... No change
Measuring weather conditions... Updated!
Reporting to MQTT topic IEEE-PyTime-IOT: {"humedad": 55, "temperatura": 24}
led apagado
Measuring weather conditions... Updated!
Reporting to MQTT topic IEEE-PyTime-IOT: {"humedad": 69, "temperatura": 25}
led apagado
Measuring weather conditions... Updated!
Reporting to MQTT topic IEEE-PyTime-IOT: {"humedad": 64, "temperatura": 25}
led apagado
```

Figura 6.17: Datos de humedad y temperatura generados por el sensor DHT11

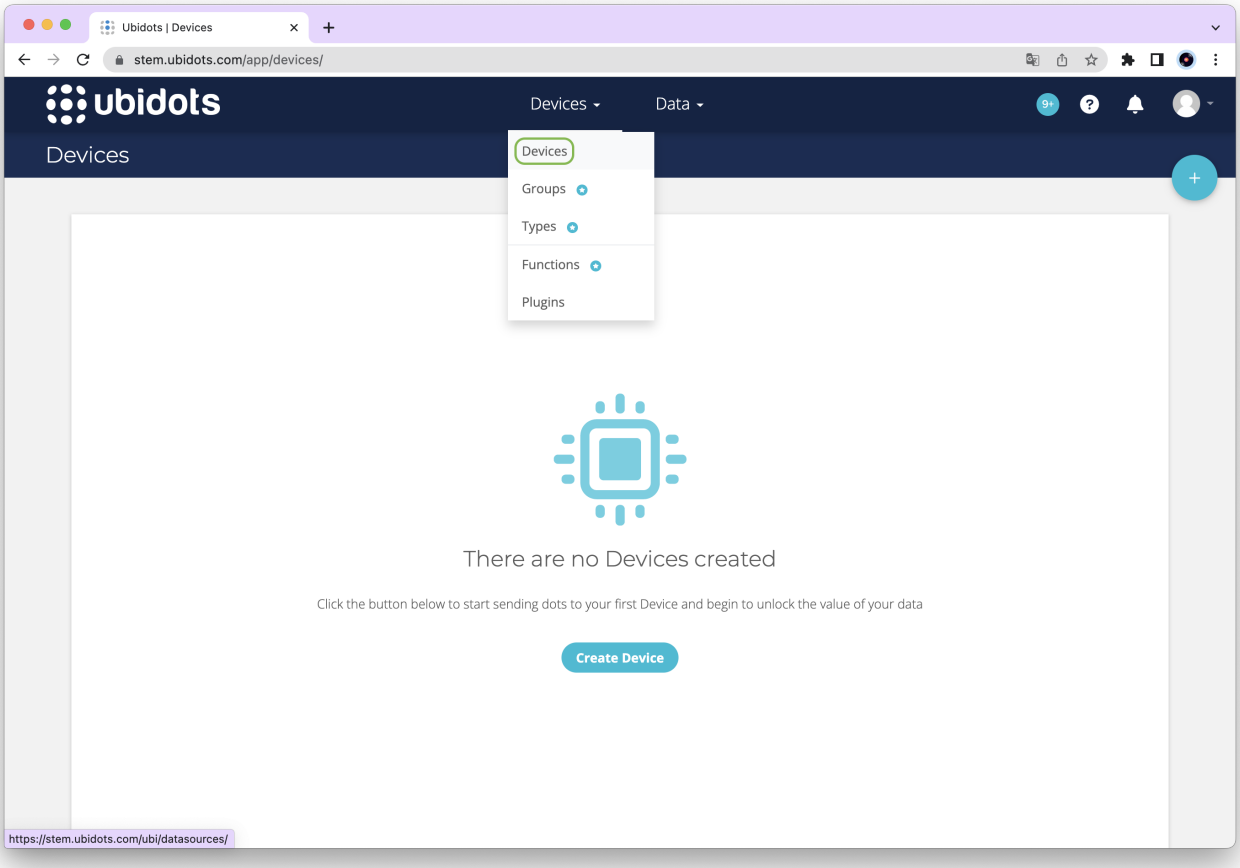

Figura 6.18: Nuevo dispositivo en Ubidots

e) En la sección de Devices escogemos el dispositivo con el nombre esp32. Luego damos un clic sobre el dispositivo y visualizamos los datos en tiempo real. Ver 6.19.

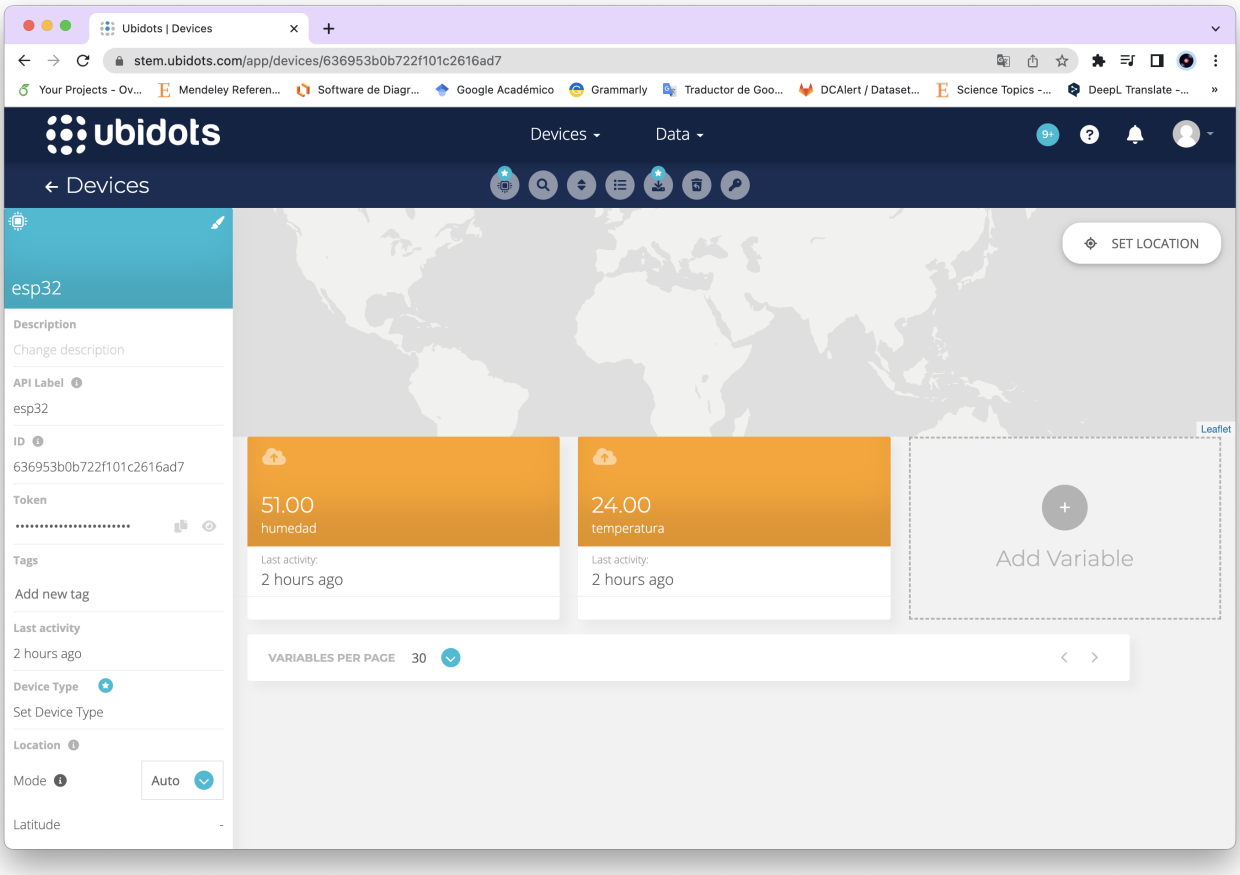

Figura 6.19: Dispositivo ESP32 en Ubidots

7 | Referencias

[1] M. Lutz, *Learning Python*, fifth edition ed. O'Reilly Media, 2013.