**Qué es un micro-controlador. El micro-controlador ESP-32**

Introducción

El entrenador QM020 utiliza la tarjeta de desarrollo ESP32 Wroom32 DevKit. Es importante definir el tipo de tarjeta de desarrollo porque hay varias y la distribución de los pines no es igual. La siguiente es el diagrama de los pies de esta tarjeta.

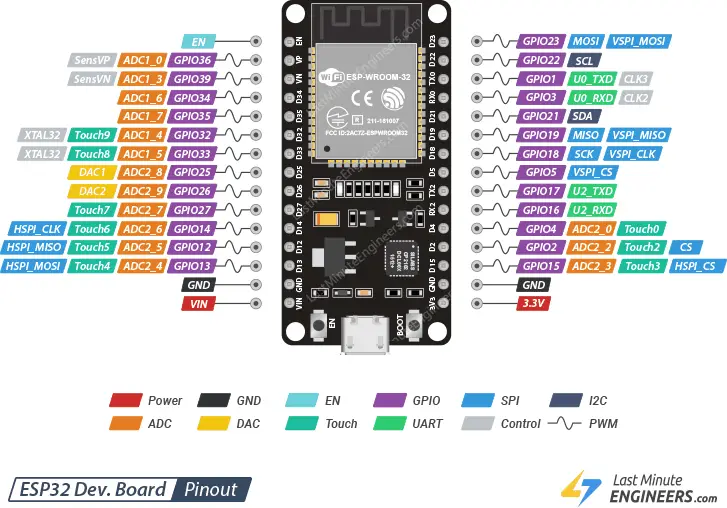
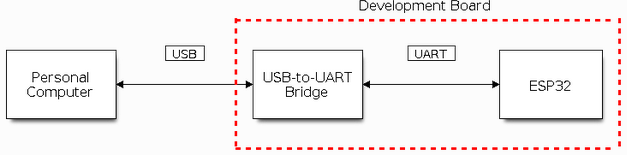
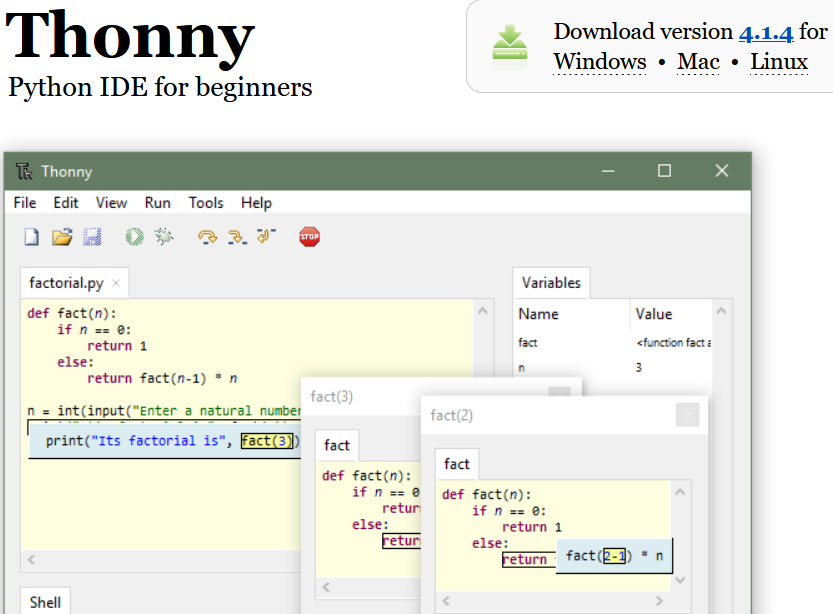


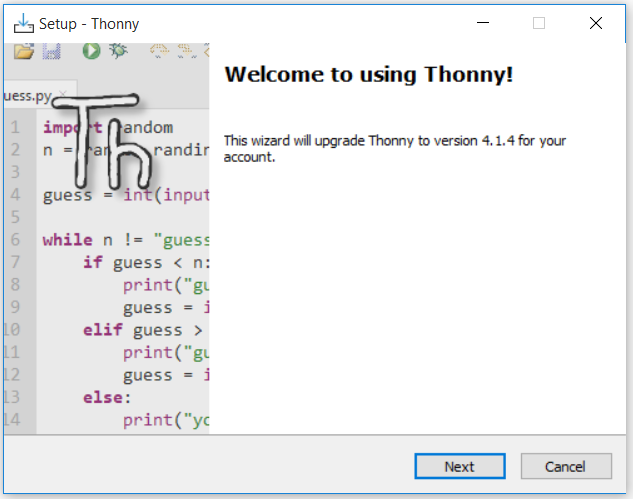
Figura 1. Distribución de pines de la tarjeta de desarrollo ESP32 Wroom32 DevKit

Para interactuar y programar el entrenador QM020, se requiere de un computador donde se debe correr un entorno de desarrollo integrado, IDE. El que utilizamos aquí es Thonny, que incluye la versión 3.7 de Python, su instalación es simple lo mismo que su utilización.



De manera que, para programar el micro-controlador ESP32 utilizando Micro-Python, se debe utilizar un computador donde corra Thonny. Para dejar Thonny listo se debe descargar de su sitio web [Thonny](https://thonny.org/). En la esquina superior izquierda de esa página aparece la versión y el sistema operativo, y simplemente se descarga.





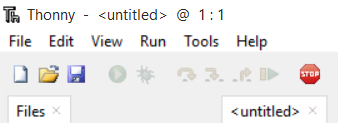
Puede haber problemas con los instaladores diciendo que por seguridad lo descargue bajo su responsabilidad, no hay ningún problema. A continuación hacer click en instalar y luego en terminar. Así queda instalado el IDE Thonny, que es con el que vamos a interactuar a lo largo de todos los experimentos.

**El firmware**

Con el fin de que se puedan correr los programas en el micro-controlador ESP32, se requiere un firmware, un programa básico que hace que el ESP32 se comporte como un pequeño computador para controlar sus funciones y sus periféricos, tales como los GPIO o pines de entrada y salida de propósito general; WiFi; Bluetooth; las entradas DAC y ADC. En el caso del entrenador QM020, el micro-controlador ya viene con un firmware, lo que permite entrar a programar directamente. Además, se pueden almacenar los programas que se van escribiendo.

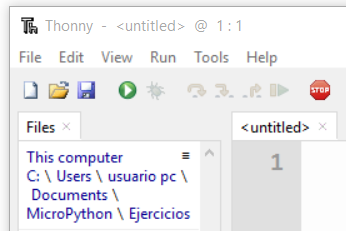
Ajustes iniciales

Con el fin de que puedan correr los programas se requieren algunos ajustes. Para ello vamos a detallar las opciones que nos ofrece:



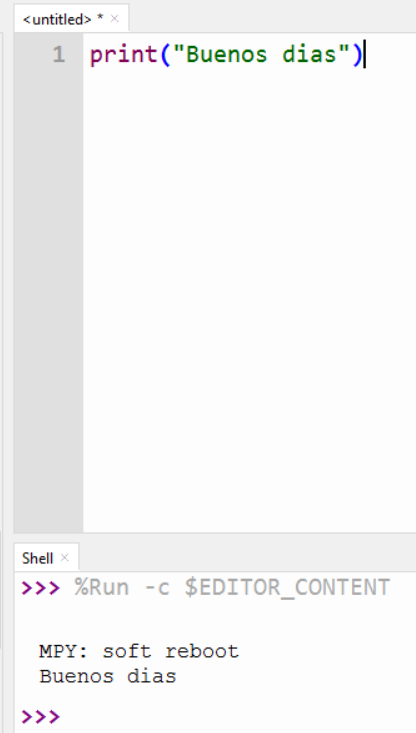
File

Hace referencia a la trayectoria en el computador de la carpeta donde va a guardar los programas o archivos con terminación .py. En el ejemplo se tiene una carpeta que se llama Ejercicios, que se encuentra en la carpeta MicroPython en Documentos. Para comenzar a escribir el primer programa le da el icono de nuevo o en File le da Nuevo. Y le presenta la siguiente pantalla:

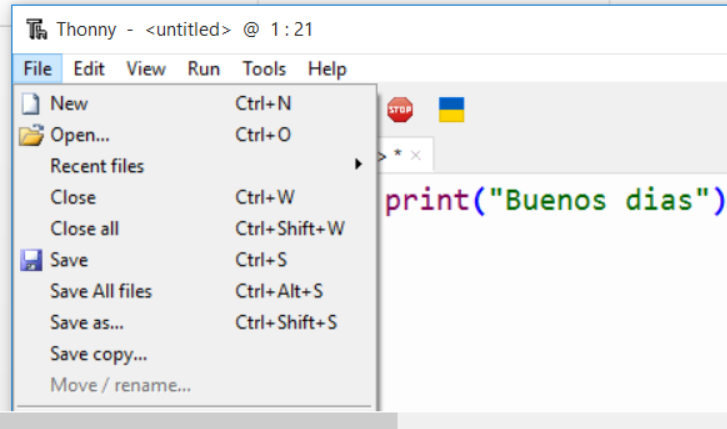


El 1, corresponde a la primera instrucción del programa que se va a escribir. 

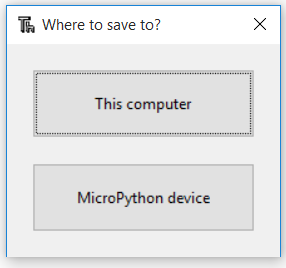
Ahora vamos a ejecutar este programa. Para esto podemos hacer click en la flecha verde o ir a Run. Vamos a hacer click en la flecha verde y luego volveremos a Run. Al hacer click, en la flecha verde, se produce la salida del programa en la zona que llamamos Consola.



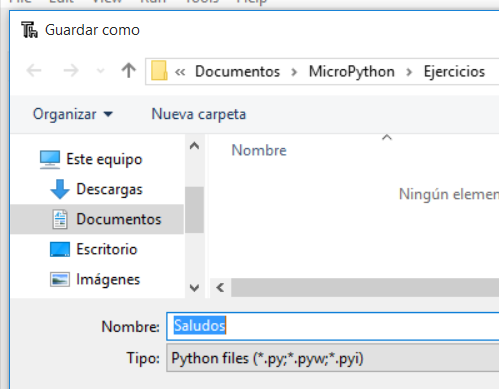
Ahora vamos a guardar el programa, hacemos click sobre Files y nos abre



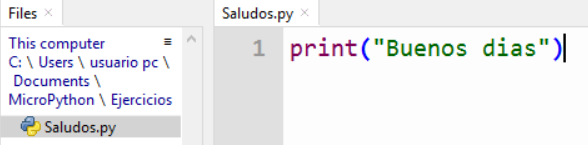
Hacemos click en Save as… Nos presenta esta pantalla, para ver donde lo va,os a guiartdar. Si en el computador o en el ESP32.



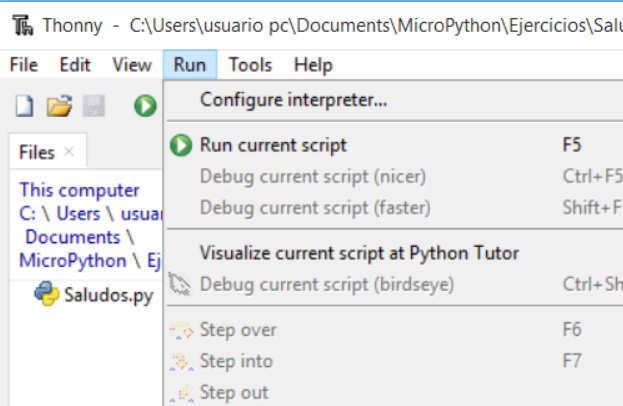
Vamos aguardarlo en el computador, lo vamos a llamar Saludos.



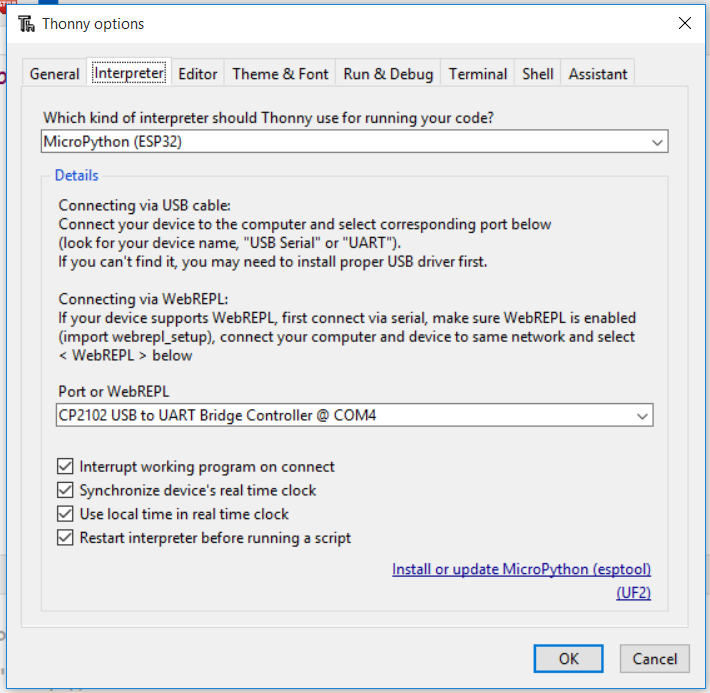
Ahora tenemos:}



Ya quedo guardado en el computador el programa Saludos.py en la carpeta Ejercicios. Ahora decíamos que se requieren algunos ajustes en ciertos casos. Veamos el cuadro



Ahora damos click en Configure interpreter, esta es la parte más importante pues se le dice a Thonny con que interpretador de Python se va a trabajar. Se le da como interpreter MicroPython(ESP32) y como puerto CP2192 USB to UART Bridge Controller @ COM4. Es decir que toma el puerto 4 del computador para hacer la transferencia al micro-controlador.



Ahora ya queda todo listo para empezar a trabajar los experimentos en MicroPython.

GPIOS o Pines

Los pines GPIO (General Purpose Input/Output), tienen un número determinado y son puertos que se pueden configurar tanto como de entradas como de salidas en los micro-controladores ESP32. Las entradas se utilizan para leer las señales externas, como sensores, interruptores o cualquier dispositivo que genere una señal eléctrica entre 0V, 3V o 5 V, que corresponden a los niveles lógicos 0 y 1; o en el caso de entradas análogas que van entre esos valores de voltaje. El mismo caso para las salidas, el micro-controlador pone a la salida de los GPIOs valores lógicos 1 o 0; o también señales analógicas entre esos voltajes.

La asignación de los pines GPIOs es muy importante, pues usar los GPIOs incorrectos puede llevar a problemas e incluso al daño del micro-controlador. De manera que sugerimos utilizar en los experimentos los siguientes pines, para los dispositivos especificxos:

# LEDS

RGB\_R = 14

RGB\_G = 12

RGB\_B = 13

CINTA\_RGB = 23

# SENSORES

HCSR04\_TRIG = 16

HCSR04\_ECHO = 4

SENSOR\_DHT11 = 5

SENSOR\_IR = 17

VS1838B = 15

FOTORESISTOR = 36

POTENCIOMETRO = 39

#LCD

LCD\_I2C\_SDA = 21

LCD\_I2C\_SLC = 22

# SONIDO

BUZZER = 19

# BOTON

BUTTON1 = 27

BUTTON2 = 18

**Programación MicroPython**

El objetivo del entrenador QMT020 es que el estudiante aprenda a programar, hacer algoritmos, de una manera sencilla, interactiva, en lenguaje MicroPython, utilizando los recursos que brinda el entrenador, cuyo componente básico es el micro-controlador ESP32. El lenguaje MicroPython es una versión ligera de Python para utilizar con los micro-controladores. Los recursos que se tienen, además del ESP32, son dispositivos de salida como LEDs y entradas como los sensores.

Un aprendizaje adicional, que se logra con el aprendizaje de MicroPython, son los fundamentos de IoT. Iot funciona con base en sensores. El sensor entrega una determinada magnitud a disposición del programa. Con base en la lectura de esa magnitud, el programa la analiza y toma decisiones, como pueden ser almacenar la lectura, presentar sus resultados en una pantalla, activar la lectura de otros sensores, o disparar una alarma, o enviar la información a través de redes a otros computadores.

Entonces, es necesario programar el micro-controlador para efectuar estas tareas, por lo que usted debe desarrollar esta destreza. Saber programar, construir algoritmos, le abre un mundo completamente nuevo de posibilidades, para crear sus propios programas, comprender lo que han hecho otros y proyectarse al mundo de la tecnología.

Por definición un computador es un dispositivo que tiene un procesador donde se ejecutan instrucciones lógicas y aritméticas; y memorias donde se almacenen los datos y las instrucciones del programa que se ejecuta. Mientras que el micro-controlador es un pequeño computador en un circuito integrado.

Desde este punto de vista el micro-controlador ESP32 es un computador, claro con poca potencia comparado con otros computadores, pero con las mismas funciones básicas, por lo que es susceptible de programar en lenguajes como C++ y Python. Lo que se escribe en estos lenguajes se conoce como lenguaje fuente, pero todos los programas que corren en los computadores lo hacen en lenguaje de máquina, es decir en 1s y 0s, que es lo que entiende el procesador del computador. Para el humano es muy complicado comprender y programar en este lenguaje por lo que se utilizan los lenguajes de programación como Java, C++ y Python.

Para convertir un programa en C++ o en Python, lenguaje fuente, a lenguaje de máquina se requiere un programa. Este programa puede ser un compilador, para el caso de C++ y muchos otros lenguajes; o un interpretador como es el caso de Python. El compilador lee todo el programa fuente y crea un programa equivalente en lenguaje de máquina. El interpretador funciona de una manera diferente, va leyendo cada instrucción del lenguaje fuente, Python, y directamente lo convierte a lenguaje de máquina y lo ejecuta.

Para correr programas Python lo más común es utilizar un IDE (Integrated Development Environment) que son plataformas de desarrollo con un software de apoyo mucho más rico. El entrenador QMT020, requiere un computador base con la plataforma Thonny, que permite la edición de los programas Python y su descarga al micro-controlador.

**Versión para uso de Thonny**

Una forma de programar el ESP32 con el firmware de MicroPython es muy útil utilizar un entorno de desarrollo integrado IDE. Para acompañar los experimentos con el entrenador QMTo20 se escogió la IDE Thonny.

Instalación de la IDE Thonny

Esta IDE se puede instalar en Windows, Mac OS X y Linux. A continuación se describen los pasos para instalas Thonny en Windows, por ser el sistema operativo más popular.

1. Ir a la página: <https://thonny.org/>
2. Descargar la versión para Windows en la sección Instructions & downloads. .
3. Correr el archivo .exe.
4. Seguir el proceso de instalación hasta su terminación.
5. Una vez terminada la instalación, abrir Thonny IDE.

pip install Cmake

La interacción se logra a través de los pines del entrenador por los cuales entran y salen las señales eléctricas del micro-controlador, con valores de 0V, 3V y 5V, que corresponden a las variables de un programa. Los pines de la tarjeta de desarrollo del entrenador se denominan GPIOs y se agrupan con base en las diferentes funciones que pueden asumir.

**Introducción a Micropython**

**Objetos en Python**

Pÿthon es un lenguaje orientado a objetos y las clases forman la base para todos los tipos de datos. La instrucción más importante en Python es la instrucción de asignación:

temperatura = 16.2

La palabra “temperatura”, Python la define como un identificador, o variable, y la asocia con el objeto que se encuentra al lado derecho del signo igual, en este caso un objeto de punto flotante con el valor “16.2”. Una foto del resultado de esa asignación lo vemos en la figura 1.

temperatura

Float

16.2

Figura 1.1. El identificador “temperatura” hace referencia a una instancia de la clase float.

**Identificadores**

Los identificadores en Python son sensibles a las letras mayúsculas y minúsculas así “Temperatura” y “temperatura” son identificadores diferentes. Los identificadores se pueden componer de casi cualquier combinación de letras y números, aunque no pueden comenzar con un número. Hay varias palabras reservadas que no se pueden utilizar como identificadores, se presentan en la Tabla 1.

|  |  |  |
| --- | --- | --- |
| **Palabras Reservadas** | | |
| And | False | Not |
| As | For | Or |
| Break | From | Pass |
| Class | Global | Raise |
| Continue | If | Return |
| Def | Import | True |
| Del | In | Try |
| Elif | Is | While |
| Else | Lambda | With |
| Except | None | Yield |

Tabla 1.1. Palabras reservadas en Python

Cada identificador esta implícitamente asociado con la “***dirección en memoria***” del objeto al cual se refiere. A un identificador, o variable, Python se le puede asignar un objeto especial, None. None es un objeto cuyo valor es nulo, es decir, no hay una dirección de memoria asociada con ese objeto.

Python es un lenguaje que no tiene una declaración previa de tipos, que asocie un identificador con un tipo de datos particular, aunque un identificador se puede asociar con cualquier tipo de objeto. En el ejemplo anterior, “16.2” se reconoce como de punto flotante y de esta manera, el identificador “temperatura” se asocia con una instancia de la clase float que tiene ese valor.

Se puede establecer un alias asignando un segundo identificador a un objeto existente:

clima = temperatura

Ahora, los dos identificadores, “temperatura” y “clima” apuntan a la dirección del mismo objeto.

temperatura

Float

16.2

clima

Figura 1.2. Los identificadores “temperatura” y “clima” son alias del mismo objeto.

Ahora, para acceder a ese objeto se puede usar cualquiera de los alias. La función Python “print” permite mostrar en pantalla el objeto por medio de la referencia.

print(clima)

print(temperatura)

16.2

16.2

Sin embargo, si se utiliza la función de asignación para crear otro objeto float, el alias continua apuntando al objeto anterior. En otras palabras la función de asignación lo que hace es crear otro objeto, perdiendo los alias.

clima = 18.5

print(clima)

print(temperatura)

18.5

16.2

**Creación y uso de objetos**

El proceso de crear una nueva instancia de un objeto se llama “***instanciación***”. En general, la sintaxis para instanciar un objeto es invocar el “***constructor***” de la clase. Por ejemplo, con la clase “Pin” se puede crear una instancia de esa clase usando una sintaxis como “pin = Pin(2,Pin.OUT)”. Así el objeto “pin” es el GPIO 2 del micro-controlador ESP32, especificado como de salida, es decir, en “p”, se toma la señal de salida de GPIO2. Otra forma de crear una nueva instancia de una clase indirectamente es llamando una función que crea y retorna tal instancia. Por ejemplo Python tiene una función llamada “sorted”, que toma una secuencia de elementos iterables como parámetro y retorna una instancia de la clase “list” que contiene los elementos ordenados.

sorted([5, 2, 3, 1, 4])

[1, 2, 3, 4, 5]

Las clases Python pueden definir uno o más métodos (o funciones miembro) que se invocan en una instancia especifica de la clase usando el operador “.”. Por ejemplo:

a = [5, 2, 3, 1, 4]

a.sort()

a

[1, 2, 3, 4, 5]

La expresión a la izquierda del punto identifica el objeto sobre el cual se va invoca el método. Cuando se usa el método de una clase, es importante comprender su comportamiento. Algunos métodos retornan información sobre el estado de un objeto pero no lo modifican, se conocen como “***accesorios***”. Otros métodos, que cambian el estado de un objeto, como el método “sort” de la clase “list”, cambian el estado de un objeto, se conocen como “***mutadores***” o “***actualizadores***”.

**Clases incluidas en Python**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Clase** | **Descripción** |  |  |  | **Immutable?** |
| bool | Valor booleano | |  |  | Si |
| int | entero (magnitud arbitraria ) | | |  | Si |
| float | Número de punto flotante | | |  | Si |
| list | secuencia mutable de objetos | | |  | No |
| tuple | secuencia inmutable de objetos | | |  | Si |
| str | cadena de caracteres | |  |  | Si |
| set | Conjunto no ordenado de objetos distintos | | | | No |
| frozenset | Forma inmutable de la clase set | | |  | Si |
| dict | Proyección asociativa de datos en forma clave-valor | | | | No |

Tabla 1.2. Clases que se utilizan bastante incluidas en Python

La tabla 1.2 presenta un resumen de clases de uso frecuente que vienen en Python; se resalta qué clases son mutables y qué clases son inmutables. A continuación se hace una breve introducción a estas clases.

**Clase Bool**

Se usa para manipular valores lógicos o booleanos, sus dos únicas instancias se expresan textualmente como True y False. El constructor “bool()” retorna False.

t = bool()

t

False

Python permite la creación de valores booleanos a partir de un tipo no booleano. La interpretación depende del tipo del parámetro. Por ejemplo, los números se evalúan False si su valor es cero o True si es diferente de cero. Veamos algunos ejemplos:

a = 2

t = bool(a)

t

True

a = 0

t = bool(a)

t

False

Las secuencias y otros tipos de contenedores, como cadenas y listas, evalúan False si están vacías y True en caso contrario.

a = [5, 2, 3, 1, 4]

t = bool(a)

t

True

a = []

t = bool(a)

t

False

**Clases int**

Estas clases son los tipos de datos primarios en Python. La clase “int” se diseñó para representar valores enteros con magnitud arbitraria. Python escoge automáticamente la representación para el entero con base en la magnitud de su valor. El constructor int() retorna el valor 0. Este constructor se puede usar para construir un objeto entero con base en un valor existente de otro tipo. Por ejemplo:

a = 3.14

t = int(a)

t

3

a = 3.99

t = int(a)

t

3

Mientras que:

a = -3.14

t = int(a)

t

-3

a = -3.99

t = int(a)

t

-3

El constructor también se puede usar para analizar gramaticalmente una cadena que se supone va a representar un valor entero, como cuando lo entra un usuario. La cadena se toma como en base 10.

s ="123"

t = int(s)

t

123

Si la cadena no tiene representación numérica, se produce un “ValueError”.

s ="Hola"

t = int(s)

t

**----------------------------------------------------------------------**

**ValueError** Traceback (most recent call last)

**Clase float**

Esta clase, en ESP32, utiliza el formato de punto flotante de 32 bits. El equivalente en punto flotante de un número entero se puede expresar directamente, como por ejemplo “2.0”, donde el cero es opcional, “2.”, para designar el mismo literal. Otra forma de literal para valores de punto flotante usa notación científica así el literal “6.022e23”, representa el valor numérico 6.022 x 1023. La forma constructor “float()” retorna “0.0”, cuando se le da un parámetro, trata de retornar el valor de punto flotante equivalente en caso contrario se produce un “ValueError”..

t = float()

t

0.0

s =2

t = float(s)

t

2.0

s ="123"

t = float(s)

t

123.0

**Secuencias list, tuple y str**

Las clases list, tuple y str, en Python son tipos de secuencias, que representan una colección de valores en las que el orden no es significativo. La clase “list” es la más general, representa una secuencia de objetos arbitrarios, como un “array” en otros lenguajes. La clase “tuple” es una versión inmutable de una lista. La clase “str” está diseñada para representar una secuencia inmutable de caracteres de texto.

**Clase List**

Una instancia de “list” almacena una secuencia de objetos. Es una estructura referencial y técnicamente almacena una secuencia de referencias a sus elementos, que pueden ser cualquier tipo de objetos incluido el objeto “None”. Los elementos de una lista se pueden indexar con un índice, que va desde *0* hasta *n-1,* si la lista es de longitud *n*. Python usa los caracteres “[ ]” como delimitadores de un literal “list”. El constructor “list()” produce una lista vacía y acepta cualquier parámetro que sea ***“iterable***”. Todos los tipos de contenedores (list, tuple, str, dictionay) son iterables.

t = list()

t

[]

a = [5, 2, 3, 1, 4]

t = list(a)

t

[5, 2, 3, 1, 4]

s ="Hola"

t = list(s)

t

['H', 'o', 'l', 'a']

s ="Perro"

t = list(s)

t

['P', 'e', 'r', 'r', 'o']

**Clase tuple**

Es una versión inmutable de una secuencia por lo que tiene una representación interna con mejor manejo de memoria que una “list”. Python usa los caracteres “()” como delimitadores de un literal “tuple”. Algo extraño, no se puede definir una “tuple” con un solo carácter, porque no es iterable.

t = tuple()

t

()

a = [5, 2, 3, 1, 4]

t = tuple(a)

t

(5, 2, 3, 1, 4)

a = {5, 2, 3, 1, 4}

t = tuple(a)

t

(1, 2, 3, 4, 5)

**Clase Str**

Está diseñada específicamente para representar una secuencia inmutable de caracteres, con base en el código de caracteres Unicode. Las cadenas tienen una representación interna más compacta que las tuplas y las lists, como se ve en la figura.

|  |  |  |  |
| --- | --- | --- | --- |
| **H** | **O** | **L** | **A** |
| **0** | **1** | **2** | **3** |

Las cadenas se pueden encerrar entre caracteres simples o comillas dobles.

a = 'Gato'

a

'Gato'

p = a[0]

p

'G'

p = a[3]

p

'o'

**Clase set**

La clase “set” representa la noción matemática de conjunto, una colección de elementos no repetidos y sin un orden implícito. La clase contiene todos los métodos de las operaciones matemáticas correspondientes. Tiene un método optimizado para verificar si un elemento específico pertenece al conjunto. Usa corchetes como delimitadores “{}”.

a = set('Hola Pythonista')

a

{' ', 'H', 'P', 'a', 'h', 'i', 'l', 'n', 'o', 's', 't', 'y'}

t = 'H' in a

t

True

t = 'H' in a

t

False

**Clase dict**

La clase Python “dict” representa un ***diccionario*** o ***proyección***, desde un conjunto de claves*, keys*, distintas a valores, *values*, asociados. Es similar a “set” pero con valores asociados.

diccionario = {"key1":<value1>,"key2":<value2>,"key3":<value3>}

hora = [8, 9, 10, 11,12]

temperatura = [14, 16, 16, 18, 20]

hora\_temp = d

{8: 14, 9: 16, 10: 16, 11: 18, 12: 20}

**Expresiones y operadores**

Vimos como los identificadores se pueden usar para identificar objetos existentes y cómo los literales y constructores se pueden utilizar para crear instancias de las clases incluidas. Se pueden combinar valores en grandes expresiones sintácticas usando una variedad de símbolos especiales y palabras claves conocidas como ***operadores***.

Las expresiones compuestas como “*a + b + c*”, que se basan en la evaluación de más de dos operaciones, el orden en que se evalúan estos operadores puede alterar el valor final de la expresión. En Python se define un orden específico de precedencia para evaluar las operaciones, en el caso de que un programador quiera alterar ese orden debe utilizar paréntesis “()”. A continuación se presentan los operadores Python comenzando con os de mayor prioridad o precedencia.

**Operadores Python**

* Concatenación de caracteres
* Lógicos o booleanos
* De igualdad
* De comparación
* Aritméticos
* A nivel de bits
* De secuencias

**Operador de concatenación de caracteres**

Para trabajar con cadenas de caracteres “str”, la operación básica es la concatenación que consiste en unir dos cadenas de caracteres, produciendo otra cadena de caracteres. El operador de concatenación +.

hola = 'Hola'

python = 'Pythonisa'

hola\_python = hola + ' ' + python # concatenamos 3 strings

print(hola\_python)

Hola Pythonista

**Operadores lógicos o booleanos**

Python soporta los siguientes operadores lógicos:

not negacion unaria (un solo operador)

and condicional “y”

or condicional “o”

x = True

y = False

x or y

True

x = True

y = False

x and y

False

x = True

not x

False

y = False

not y

True

**Operadores de igualdad**

Python soporta los siguientes operadores para probar dos conceptos de igualdad:

is la misma identidad

is not identidad diferente

“==” equivalente

!= no equivalente

La expresión *a is b* se evalúa “*True*”, cuando los identificadores *a* y *b* son alias. La expresión *a == b,* prueba una noción más general de equivalencia. Si los indicadores se refieren al mismo objeto, entonces se evalúa “*True*”. También *a == b,* se evalúa “*True*”, cuando los identificadores se refieren a diferentes objetos pero tienen valores que se consideran diferentes.

a = 3

b = a

a == b

True

a = 3

b = 3

a == b

True

**Operadores de comparación**

Los tipos de datos pueden definir un orden natural vía los siguientes operadores:

< menor que

<= menor o igual

> mayor que

>= mayor o igual

Para los tipos de datos numéricos tiene el comportamiento esperado y para las cadenas “str”, están definidos lexicográficamente y son sensibles a letras mayúsculas y minúsculas.

**Operadores aritméticos**

Python soporta los siguientes operadores aritméticos:

\*\* exponenciación

% operador módulo

// división entera

/ verdadera división

\* multiplicación

+ suma

* Resta

La exponenciación es el operador de más alta prioridad, eleva el operando de la izquierda a la potencia del operador de la derecha. Python tiene una librería *math*, que incluye la función pow(), similara a \*\*.La siguiente prioridad la tienen los operadores: “%, //, /, \*” que tienen entre ellos la misma prioridad. Los de menor prioridad son los operadores “+, -“ que tienen la misma prioridad

El uso de la suma, resta y multiplicación es directo, observando que si ambos operadores son “*int*”, entonces el resultado también es de tipo “*int*”; si uno solo o ambos operadores son de tipo “*float*”, el resultado será un “*float*”.

En Python, el operador “/” designa una operación verdadera, retornando el resultado del cálculo en punto flotante. También incluye el operador “//”, cuyo resultado es el piso de la división, si en la expresión a // b, ambos operadores son “int”, el resultado será de tipo “int”, o si alguno o ambos son “float”, el tipo del resultado será “float”. Se debe tener en cuenta que cuando alguno de los operadores es negativo, se conserva el resultado piso.

a = 4

b = 3

a / b

1.3333333333333333

a = 4

b = 3

a // b

1

a = 8

b = 3

a // b

2

a = 8

b = 3.0

a // b

2.0

a = -8

b = 3

a // b

-3

a = 8

b = -3.0

a // b

-3.0

La expresión *a % b* devuelve el residuo de dividir el primer operador *a* por el segundo operador *b*. El operador % se utiliza con operadores numéricos: enteros y flotantes, retornando un valor entero o flotante.

a = 7

b = 3

a % b

1

a = 7.

b = 3.

a % b

1.0

**Operadores a nivel de bit**

Python provee las siguientes operaciones bit a bit para enteros.

~ complemento bit a bit

& and bit a bit

| or bit a bit

^ or exckusivo bit a bit

<< desplaza bits a la izquierda rellenando de ceros

>> desplaza bits a la derecha rellenando de ceros

x = 2 # Entero 2 en bits es 000010

y = 7 # Entero 7 en bits es 000111

x & y

2

x = 2 # Entero 2 en bits es 000010

y = 7 # Entero 7 en bits es 000111

x | y

7

x = 2 # Entero 2 en bits es 000010

x << 2

8

x = 2 # Entero 2 en bits es 000010

x >> 1

1

**Operadores de secuencia**s

Cada uno de los tipos de secuencia incluidos en Python “(Str, tuple y list)” soportan las siguientes instancias del operador:

s[j] Elemento en el índice j,

s[start:stop] Tajada que incluye los índices start y stop

s[start:stop:step] Tajada que incluye los índices [start, start + step]

[start, start +2 step] … [start: stop]

s + t Concatenacion de dos cadenas

K \* s Corto para s + s + s + + … (k veces)

val **in** s Verifica contenido

val **not in** s verifica no contenido

Python se basa en secuencias con índice 0, así que una secuencia de longitud *n* tiene los elementos indexados de *0* a *n-1.* Los índices pueden ser negativos, lo que denota un elemento a una distancia desde el final de la secuencia hacia atrás.

a = [5, 2, 3, 1, 4]

b = [6, 9, 8, 7, 10]

c = a + b

c

[5, 2, 3, 1, 4, 6, 9, 8, 7, 10]

c[4:8]

[4, 6, 9, 8]

c[2:10:2]

[3, 4, 9, 7]

c[-3]

8

10 in c

True

12 in c

False

Todas las secuencias definen operaciones de comparación basada en el orden lexicográfico, efectuando la comparación elemento por elemento, hasta encontrar la diferencia, si es que se presenta.

s==t Equivalencia elemento por elemento

s != t No equivalente

s < t Lexicográficamente menor que

s <= t Lexicográficamente menor que o igual a

s > t Lexicográficamente mayor que

s >= t Lexicográficamente mayor que o igual a

**Precedencia de los operadores**

Para que Python evalúe las expresiones, se requieren reglas claras, como por ejemplo evaluar la expresión 4 + 3 \* 2. El orden formal de precedencia para los operadores Python se da en la tabla 1.4. Los operadores con más alta precedencia se evaluarán primero que aquellos con más baja prioridad, a no ser que la expresión contenga paréntesis, que se evalúan primero.

|  |  |  |
| --- | --- | --- |
| **Precedencia de los operadores** | | |
| **Pr.** | **Tipo** | **Símbolos** |
| 16 | llamadas a funciones/métodos | expr(…) |
| 15 | contenedor subíndice/tajada | expr[…] |
| 14 | exponenciación | \*\* |
| 13 | operadores unarios | +expr, -expr, ~expr |
| 12 | multiplicación, división | \*, /, //, % |
| 11 | suma, resta | +, - |
| 10 | desplazamiento bit a bit | <<, >> |
| 10 | and bit a bit | & |
| 9 | xor bit a bit | ^ |
| 8 | or bit a bit | | |
| 7 | comparaciones | is, is not, ==, !=, <, <=, >, >= |
| 6 | contenido | in, not in |
| 5 | not lógico | not expr |
| 4 | and lógico | and |
| 3 | or lógico | or |
| 2 | condicional | val1 if condition else val2 |
| 1 | asignaciones | =, +=, -=, \*=, etc. |

Tabla 1.2. Precedencia de los operadores en Python, de la precedencia más alta a la más baja. El litera “expr”, es un identificador o el resultado de una expresión evaluada previamente.

a = 4 + 3 \* 2

a

10

a = (4 + 3) \* 2

a

14

**Control de flujo**

Flujo en la ejecución de un programa de computador, hace referencia al orden en que se ejecutan las instrucciones que conforman el programa. Todo lenguaje de programación de computadores se compone de tres estructuras básicas: Secuencia, Decisión y Ciclos.

1. **Secuencia**. Es una secuencia de instrucciones, o bloque de instrucciones. Las instrucciones del bloque se ejecutan secuencialmente desde la primera hasta la última instrucción.
2. **Decisión.** Es una estructura que le permite al programa tomar decisiones, cambiar el flujo de ejecución, si se cumple una condición. Si se cumple una condición, se ejecuta un bloque de instrucciones. Si no se cumple, se ejecuta otro bloque de instrucciones. Una vez terminada la ejecución de uno u otro bloque de instrucciones, continúa la ejecución del programa con la siguiente instrucción en la secuencia.
3. **Ciclos.** Se ejecuta un bloque de instrucciones varias veces hasta cuando se cumple una determinada condición. Una vez que se cumple la condición se sale del ciclo y continúa la ejecución de la siguiente instrucción en la secuencia. En caso de que la condición esté mal programada, el bloque de instrucciones se ejecuta infinitamente.

**Indentacion**

La indentación hace referencia a los espacios que se dejan a la izquierda de una línea de código. Todos los lenguajes de programación modernos utilizan la indentación, como C++, Java, Javascript y claro, Python; para delimitar los bloques de instrucciones en el control de flujo. Python se basa en la indentación para designar la extensión de un bloque, o cualquiera de los bloques anidados dentro de él. Por esto, se debe usar el mismo número de espacios para todas las instrucciones que pertenecen a un bloque o el interpretador va a generar errores.

**Estructuras de control de flujo en Python**

Hay una sintaxis común a todas las estructuras de control en Python, para definir los bloques de código. El carácter “:” se usa para delimitar el comienzo de un bloque de código, que actúa como un cuerpo en una estructura de control. Ese cuerpo de código se coloca a la derecha de “:”, de forma ***indentada***, lo que define la prioridad de ejecución de ese bloque.

**Estructura de decisión.**

If <condición>:

<bloque de código 1>

else:

<bloque de código 2>

La instrucción de “<condición>” que acompaña el **if**, se conoce como una instrucción relacional, ya que hace la relación entre dos variables, o entre una variable y una constante. La relación se hace con base en un operador de condición, cuyo resultado es “True” (verdadero) o “False” (falso). Si el resultado de la instrucción relacional es “True”, se ejecuta el “<bloque de código 1>”, si el resultado es “False”, el control de secuencia se va por el **else** y se ejecuta el “<bloque de código 2>”. Una vez se ejecuta cualquiera de los bloques de código, el programa sale de esa estructura.

Por ejemplo el código Python pregunta si el estudiante obtuvo una nota mayor o igual 60, si es así, aprueba la asignatura, sino la pierde.

if nota >= 60:

print('Aprobado')

else:

print('Suspendido')

En este caso la variable “nota” se relaciona a través del operador “>=” con la constante 60. Dependiendo del contenido de la dirección de memoria a la que apunta la variable ”nota”, el resultado de esa instrucción relacional será “True” o “False”.

Si el contenido de esa dirección de memoria es 60 o mayor, entonces el resultado de la relación es “True”, se imprime “Aprobado” y se sale de la estructura. En caso de que la dirección de la variable “nota” sea menor de 60, el resultado de la instrucción relacional es “False”, se imprime “Suspendido” y se sale de la estructura.

Otra forma de estructura condicional que permite personalizar con más detalle la decisión, incluye otra instrucción relacional con la palabra clave **elsif.** En este caso, cuando no se cumple la “<condición 1>”, entra a ejecutar la “<condición 2>”. Si el resultado de la “<condición 2>”, es “True”, se ejecuta el “<bloque de código 2>” y sale de la estructura; si el resultado es “False”, se ejecuta el <bloque de código 3>

If <condición 1>:

<bloque de código 1>

elif <condición 2>:

<bloque de código 2>

else:

<bloque de código 3>

Veamos un ejemplo en código en Python donde se lee la temperatura ambiente de un sensor y es de 25 grados, se sabe que la temperatura máxima para un proceso es de 35 grados. Si la temperatura es mayor que la máxima, se puede pensar en activar un ventilador, en este caso imprime "temperatura muy alta". Si la temperatura no es mayor, es posible que sea igual a la máxima, entonces se imprime como una advertencia "temperatura alta". Pero si es menor a la temperatura máxima se imprime "temperatura normal"-

temperatura = 25

maxima = 35

if temperatura > maxima:

print("temperatura muy alta")

elif temperatura == maxima:

print("temperatura alta")

else:

print("temperatura normal")

temperatura normal

Al observar la indentación de un programa, se puede ver la estructura del programa, sus bloques de instrucciones, dependiendo de las decisiones que se toman. En Python el no hacer bien la indentación implica que el programa no funciona como esperamos o no se ejecuta.

Operadores de relación más comunes en Python:

|  |  |  |
| --- | --- | --- |
| Operador | Nombre | Ejemplo |
| == | Igual | x == y |
| != | Diferente | x != y |
| > | Mayor | x > y |
| < | Menor | x < y |
| >= | Mayor o igual | x >= y |
| <= | Menor o igual | x <= y |

**Estructuras de ciclos o iterativas**

Estas estructuras son bastante poderosas pues permiten la ejecución de un bloque de código varias veces. Hay dos estructuras en Python que permiten la ejecución de ciclos: la estructura **for** y la estructura **while**.

**Ciclo for**

Los ciclos **for** se basan en índices con los que se recorre una estructura de datos iterable. Es muy simple, sin embargo, tiene limitaciones porque no sabemos dónde reside el elemento determinado en la secuencia. En la forma más simple, la sintaxis ***range(n)*** genera la serie de índices válidos para recorrer la secuencia. Un iterable puede ser una lista, una tupla, un diccionario, la secuencia que genera una función ***range***, un archivo. La clase *range*, devuelve un iterable cuyos valores van de *0* a un *n -1*.

Sintaxis básica:

for <variable\_del\_ciclo> in <iterable>:

<bloque de código>

nums = [4, 22, 9, 84]

for n in nums:

print(n)

4

22

9

84

for i in range(5):

print(i)

0

1

2

3

4

**Ciclo while**

Se utiliza para ejecutar un bloque de código repetidamente, mientras se cumpla una condición. Su sintaxis es la siguiente:

While condición:

<bloque de código>

Como en la estructura **if**, la condición es un instrucción relacional que se evalúa True o False, si es True, se ejecute el bloque de código, en caso contrario se sale de la estructura.

numero = 1

print('Tabla del 9')

while numero <= 10:

c = numero \* 9

print(numero)

numero += 1

print('Fin')

Tabla del 9

1

2

3

4

5

6

7

8

9

10

Fin

**Instrucción break**

La instrucción ***break***, interrumpe inmediatamente la ejecución de un ciclo **while** o de un ciclo **for**, cuando se ejecuta dentro del bloque de instrucciones.

a = [5, 2, 3, 1, 4]

busca = 3

for item in a:

print(item)

if item == busca:

break

5

2

3

**Funciones**

Se usa el término función para describir una función tradicional incorporadas que se invoca sin el contexto de una clase particular o de la instancia de una clase, tal como ***sorted(datos)*.** Se usa el término más específico ***método*** para describir una función que pertenece a una clase, que se invoca bajo un objeto específico usando una sintaxis de paso de mensajes orientada a objetos, tal como: ***datos.sort().*** Para mostrar la sintaxis que definen funciones, veamos la siguiente función que cuenta el número de elementos repetidos en una estructura iterable.

def cuenta(datos,busca):

n = 0

for item in datos:

if item == busca:

n += 1

return n

La primera línea comienza con la palabra clave ***def***, luego el nombre de la función, en este caso **cuenta**; y se establece el número de parámetros que espera, así como los nombres que identifican esos parámetros, en este caso **datos** y **busca,** terminando con el delimitador **“:”.** El resto es el bloque de instrucciones de la función, que se expresa como un bloque indentado. Dentro del bloque de la función se usa la instrucción ***return***, que indica la terminación de la función y que el valor expresado, en este caso **n**, se debe retornar al programa que ha llamado la función.

def contiene(datos,busca):

for item in datos:

if item == busca:

return True

return False

a = (1,4,8,0,2,4,)

cuenta(a,4)

2

cuenta(a,6)

0

contiene(a,4)

True

contiene(a,6)

False

**Recursión**

En Python una función puede llamar a otras funciones. Incluso es posible que una función pueda llamarse a sí misma. A este tipo de construcciones se les conoce como funciones recursivas. La recursividad permite que un bloquede instrucciones se ejecute varias veces.

Veamos el ejemplo siguiente:

def imprimir(x):

if x>0:

print(x)

imprimir(x-1)

imprimir(4)

En este caso se va llamar 4 veces la misma función hasta cuando se cumpla la condición. Siempre se debe tener en cuenta que dentro del cuerpo de la función se debe variar la variable de control que en el ejemplo es la x. Para cada llamado de la función se establecen una serie de pilas con el contenido de las variables y las posiciones de memoria donde se van almacenando los bloques de código, por lo que se debe ser muy cuidadosa cuando se utilicen funciones recursivas. Si la variable de control no varía, es posible que la función se trate de ejecutar infinitamente y la llamada recursiva se tome toda la memoria del micro-controlador llevando a una falla.

**EXPERIMENTOS CON EL MICRO-CONTROLADOR ESP32**

En esta parte del manual, tal vez la más importante, es donde vamos a trabajar el desarrollo de algoritmos en Micro-python. Con el fin de tener mayor claridad sobre la parte del manual correspondiente a Micropython, vamos a comenzar los experimentos, corriendo los programas que se usaron para la explicación de esos temas.

Para todos los siguientes experimentos ya se debe tener listo todo el montaje requerido para utilizar el micro-controlador ESP36 y en él se ejecuten todos los programas correspondientes a los experimentos que se detallan a continuación. La salida de los resultados de estos programas se presentan en la consola de Thonny.

**Experimento 1.** Primeros programas

**Objetivo**. Correr los primeros programas que se presentaron en la parte de conceptos básicos de Micro-Python para afianzar los conocimientos. Ver cómo se presenta los resultados de estas salidas.

**Procedimiento.**

**1.** Ejecutar el siguiente código

# El identificador o variable “temperatura” apunta al objeto flotante 16.2

temperatura = 16.2

# Ambas variables apuntan al mismo objeto

clima = temperatura

# La función “print”, presenta la salida el objeto al que hace referencia la variable que va en ()

print(clima)

print(temperatura)

Ahora, ejecute:

clima = 18.5

print(clima)

print(temperatura)

**2.** Ejecutar el siguiente código

# La función “sorted”, es una función incorporada que ordena la lista que viene en los ().

sorted([5, 2, 3, 1, 4])

Ahora ejecute el siguiente código

# Aquí el método “lista.sorted”, que ordena la lista

a = [5, 2, 3, 1, 4]

a.sort()

a

**3.** Ejecute los siguientes códigos

# Uso del constructor boo().

t = bool()

a = 2

t = bool(a)

t

a = 0

t = bool(a)

t

**4.** Ejecute los siguientes códigos

# Uso de la clase int()

a = 3.14

t = int(a)

t

a = -3.14

t = int(a)

t

s ="123"

t = int(s)

t

s ="Hola"

t = int(s)

**5.** Ejecute los siguientes códigos

# Uso de la clase float()

t = float()

t

s =2

t = float(s)

t

s ="123"

t = float(s)

t

**6.** Ejecute los siguientes códigos

# Uso de la clase list()

a = [5, 2, 3, 1, 4]

t = list(a)

s ="Hola"

t = list(s)

t

s ="Perro"

t = list(s)

t

**7.** Ejecute los siguientes códigos

# Uso de la clase tuple()

t = tuple()

t

a = [5, 2, 3, 1, 4]

t = tuple(a)

t

a = {5, 2, 3, 1, 4}

t = tuple(a)

t

**8.** Ejecute los siguientes códigos

# Uso del tipo de dato str, el más importante en Python

a = 'Gato'

a

p = a[0]

p

p = a[3]

p

**9.** Ejecute los siguientes códigos

# Uso de la clase set()

a = set('Hola Pythonisa')

a

t = 'H' in a

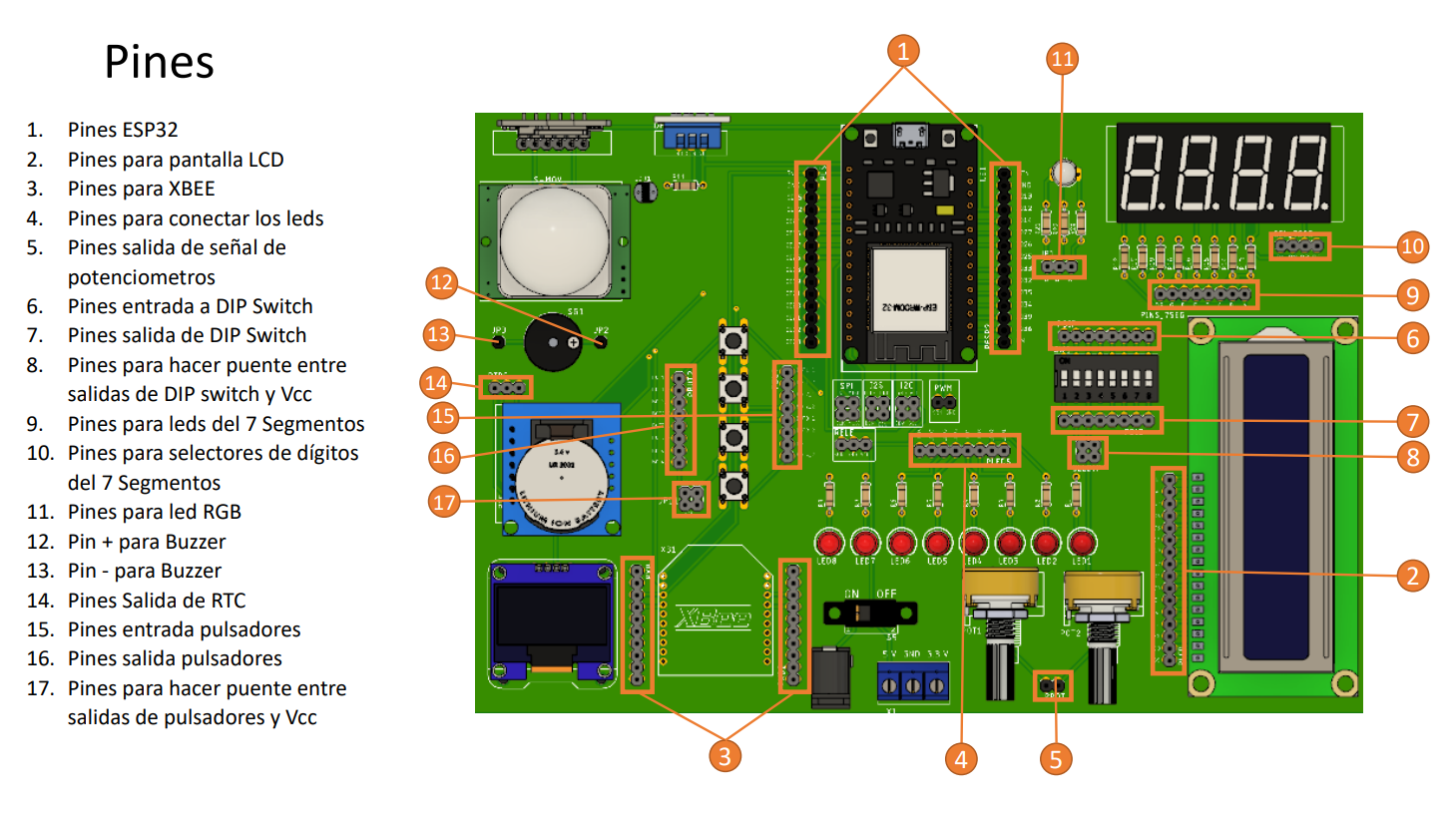
t

t = 'H' in a

t

**Experimento 2**. Reconocimiento del equipo.

**Objetivo.** Aprender a reconocer los diferentes componentes del equipo, su alimentación y otros. En la siguiente grafica se observa la distribución de los componentes del entrenador, con el fin de que el estudiante reconozca los diferentes componentes y la forma en que se conectan para la realización de los experimentos.



**Procedimiento**

1. Sin conectar el equipo, reconozca cada uno de los componentes del entrenador QTM020.

2. Conecte el micro-controlador ESP32 al servidor con el cable USB- C2.

3. Conecte el equipo, con el conector USB-C3

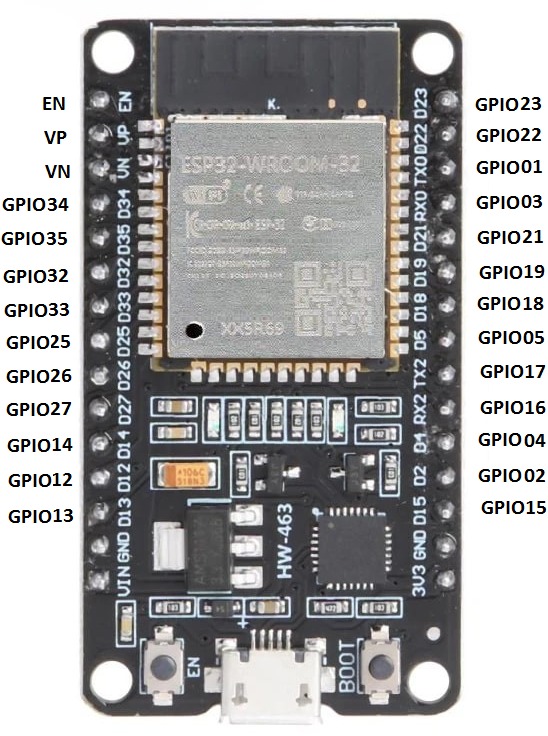
4. Coloque el switch en ON, observe en el microcontrolador, que permanece prendido un LED rojo y a veces se prende un LED azul.

En el servidor, cree una carpeta sugerimos el nombre Micropython, donde se van a almcenar todos los programas-

**Experimento 3.** Conexión de los pines GPIOs del microcontrolador.

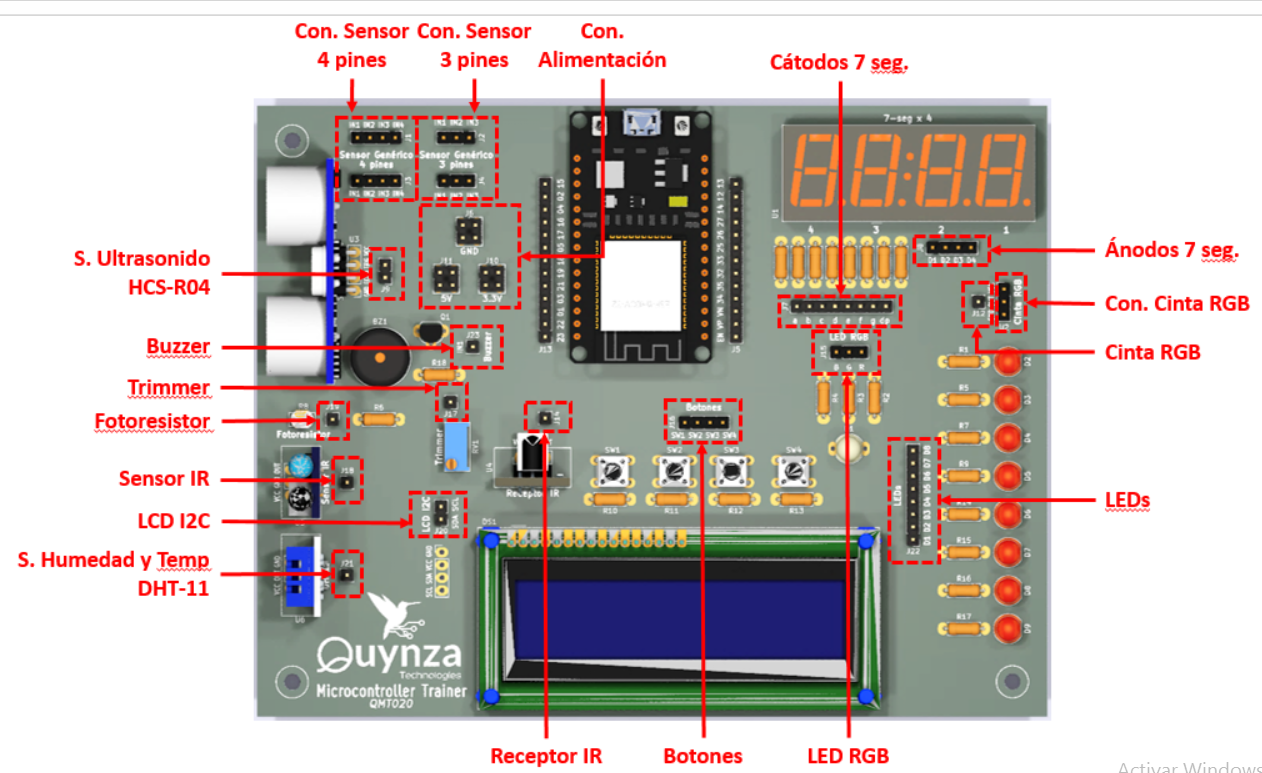
El entrenador MQT020 ya viene con varias conexiones alambradas a los diferentes dispositivos. Esto es de gran ayuda en el desarrollo de los experimentos, eliminando problemas de conexión y posibles daños a los componentes, disminuyendo el esfuerzo de realizar las conexiones y el tiempo dedicado a hacerlo.

Es por esto que solo aparecen 26 pines del micro-controlador, que tiene 30, no aparecen los correspondientes a los pines VIN, GND y 3.3V. ¡!Es importante tener en cuenta este detalle para identificar apropiadamente los GPIOs!!





De esta manera, el micro-controlador recibe o envía señales dependiendo del tipo de componente. Los LEDs, el zumbador, la pantalla, los controla el micro-controlador enviando señales desde los GPIOs a los pines asociados a cada uno de esos componentes. Mientras que el micro-controlador recibe por sus GPIOs las señales que envían los sensores, la fotoresistencia y el



1. Conexión sensor de 4 pines. Son dos conexiones de 4 pines para sensores que trabajan con señales. Le da mucha flexibilidad al entrenador para trabajar con otros sensores que tengan 4 señales.

2. Conexión sensor de 3 pines. Son dos conexiones de 3 pines para sensores que trabajan con señales. Le da mucha flexibilidad al entrenador para trabajar con otros sensores que tengan 4 señales.

3. Conexiones de alimentación.

4. Cátodos de 7 segmentos. Son 8 pines para conectarse a los cátodos de los displays de 7 segmentos.

5. Ánodos de 7 segmentos. Son 4 pines para conectarse a los ánodos de los displays de 7 segmentos.

6. Conexión para la entrada de los 3 cables de la cinta LED.

7. Señal de control para la cinta LED. Son 3 pines.

8. Conexiones a los 8 LEDs. Son 8 pines para el manejo de cada uno de los 8 LEDs.

9. Conexiones al LED RGB. Son 3 pines para controlar los tres colores, rojo, verde y azul.

10. Conexiones para el manejo de los botones, uno para cada botón.

11. Receptor de Infra-rojo. 1 pin para la entrada de la señal.

12. Sensor de humedad y temperatura. 1 pin, para la salida de la señal.

13. Señal I2C para la salida por pantalla 2 pines.

14. Fotoresistencia, 1 pin de salida.

15. Reóstato o trimmer, 1 pin de entrada.

16. Zumbador, 1 pin de salida.

17. Sensor de ultrasonido, 2 pines de salida.

GPIOs utilizados

|  |
| --- |
| 4 |
| 5 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 17 |
| 18 |
| 19 |
| 21 |
| 22 |
| 23 |
| 27 |
| 36 |
| 39 |

**Experimentos con actuadores o dispositivos de salida**

En estos experimentos se utilizan los LEDs de colores, el LED RGD, la cinta LED RGB, la pantalla LCD1602 y los cuatro displays de 7 segmentos.

**Experimento 4.** Prender y apagar el LED azul del micro-controlador ESP32.

**Objetivo.** Aprender a codificar un programa en micropython con el IDE Thonny y correrlo en el ambiente del entrenador QMT020. Aprender el uso de las librerías **time** y **machine**.

**Procedimiento**

Conectar el cable de alimentación de energía al microcontrolador con el USB-C3. Conectar el servidor con el microcontrolador a través del cable USB-C2.

Abrir en el servidor el IDE Thonny, dejar listo para escribir el siguiente programa:

El módulo **time** de la **librería** estándar de **MicroPython** proporciona un conjunto de funciones para indicar el tiempo que el microcontrolador lleva encendido, medir intervalos de tiempo e introducir retardos (esperas) en la ejecución de los programas, el tiempo y la fecha del día.

El módulo **machine** contiene funciones específicas relacionadas el hardware que se utiliza en una base especifica. En este caso el entrenador funciona con la base ESP32-S2-WROOM, ver figura 1. La mayoría de estas funciones tiene que ver con el acceso directo y sin restricciones al control de bloques de hardware en el sistema. Su uso incorrecto puede llevar incluso al daño del microcontrolador.

Con la instrucción “led=Pin(2, Pin.OUT)”, se crea un objeto Pin con nombre led, con una señal discreta que sale por el GPID 2 del micro-controlador. Con la instrucción “led.value(1)”, sale por el GPIOD 2, una señal constante, voltaje de 5 volts, que hace que el LED se encienda. Con la instrucción “led.value(0)” sale por el GPID 2, una señal constante de 0 voltios, que hace que se apague el LED. Cómo es un LED propio del micro-controlador, el GPIO2 no se vuelve a utilizar.

**1.** Corra el siguiente programa

LED\_Azul

# Importa las librerías requeridas

import time

from machine import Pin

# Crea un objeto pin como LED de salida

led=Pin(2, Pin.OUT)

led.value(1) # Enciende el LED

time.sleep(1.0)

led.value(0) # Apaga el LED

Cambie el tiempo en la instrucción time.sleep(1.0) y observe lo que sucede.

**2.** Pero ahora, vamos aprender y a apagar el LED azul durante determinado tiempo.

LED\_AzulIntermitente

# Importa las librerías requeridas

import time

from machine import Pin

# Crea un objeto pin como LED de salida

led=Pin(2, Pin.OUT)

while True:

led.value(1) # Enciende el LED

time.sleep(0.25)

led.value(0) # Apaga el LED

time.sleep(0.25)

Aquí se utiliza la estructura de control de flujo en la forma while True:. Es decir que siempre se va a estar ejecutando el bloque de instrucciones que sigue al while. Observe la indentación del bloque de instrucciones del while.

Ahora corra el programa y ejecútelo varias veces cambiando los intervalos de espera.

Con la instrucción “ time.sleep(1.5)”, se demora la ejecución de la siguiente instrucción 1.5 segundos.

2. Escribir en el editor el siguiente programa:

import time

from machine import Pin

led=Pin(2, Pin.OUT) #Define el pin led as salida

while True:

led.value(1) # Enciende la luz del LED

time.sleep(1.5)

led.value(0) # Apaga lla luz del LED

time.sleep(1.5)

Observe la ejecución del programa y vea el tiempo de prendido t apagado del LED.

Disminuya el tiempo de parpadeo del LED.

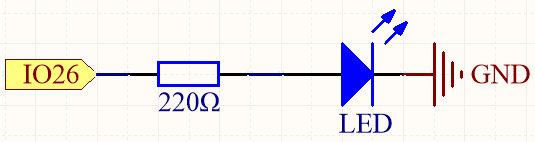
Aumente el tiempo de parpadeo del LED.

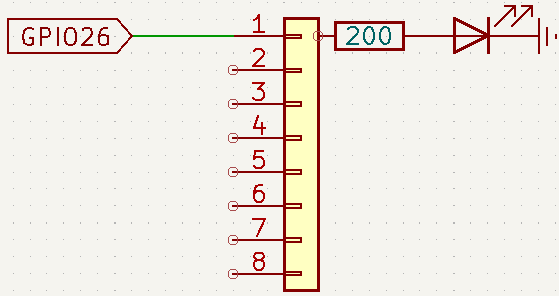
Guarde el programa en la carpeta

Guarde el programa en el microcontrolador.

**Experimento 5.** Trabajar con el LED con uno de los LEDs externos.

Ahora edite el programa anterior y cambie el pin de salida por el 26, “led=Pin(26, Pin.OUT)”. Corresponde al siguiente circuito:





Titila0

import time

from machine import Pin

led=Pin(26, Pin.OUT) #Define el pin led as salida

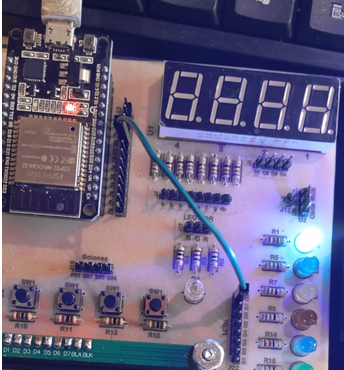
while True:

led.value(1) # Enciende la luz del LED

time.sleep(1.5)

led.value(0) # Apaga lla luz del LED

time.sleep(1.5)



Forma de conexión con el ESP32. Se toma el pin correspondiente al pin 26 y se lleva a un LED de salida.

Observe la ejecución del programa y vea el tiempo de prendido t apagado del LED.

Disminuya el tiempo de parpadeo del LED.

Aumente el tiempo de parpadeo del LED.

Guarde el programa en la carpeta

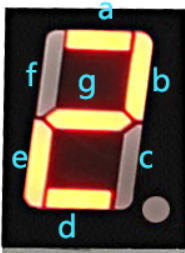
Guarde el programa en el micro-controlador.

**6.** Utilización del display de 7 segmentos

**Objetivo**. Con base en el método radint de la random, se toma un número aleatorio entre 0 y 9; y se muestra en el Display de 7 segmentos

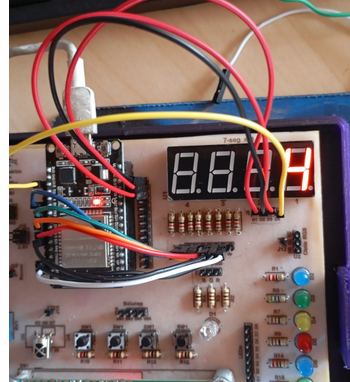
Los LEDS de 7 segmentos, permiten, a través de la iluminación de ciertos segmentos mostrar los números decimales de 0 a 10. En el caso de la figura se tiene iluminados los segmentos a, b, g y d. que corresponden al binario 1011011 de la tabla.

josemiguel1886



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **a** | **b** | **c** | **d** | **e** | **f** | **g** | **Digito** |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 9 |

**Procedimiento** Con base en el método radint de la random, se toma un número aleatorio entre 0 y 9; y se muestra en el Display de 7 segmentos. Correr el siguiente código.



from machine import Pin

import time

import random

# Para la selección del display.

d1 = Pin(12,Pin.OUT)

d2 = Pin(13,Pin.OUT)

d3 = Pin(14,Pin.OUT)

d4 = Pin(15,Pin.OUT)

# Se va trabajar con el disply 1

d1.on()

d2.off()

d3.off()

d4.off()

numero = random.randint(0, 9)

leda = Pin(16,Pin.OUT)

ledb = Pin(17,Pin.OUT)

ledc = Pin(18,Pin.OUT)

ledd = Pin(19,Pin.OUT)

lede = Pin(21,Pin.OUT)

ledf = Pin(22,Pin.OUT)

ledg = Pin(23,Pin.OUT)

while True:

if numero == 0:

leda.off() #1

ledb.off() #1

ledc.off() #1

ledd.off() #1

lede.off() #1

ledf.off() #1

ledg.on() #0

print("zero")

if numero == 1:

leda.on() #0

ledb.off() #1

ledc.off() #1

ledd.on() #0

lede.on() #0

ledf.on() #0

ledg.on() #0

if numero == 2:

leda.off() #1

ledb.off() #1

ledc.on() #0

ledd.off() #1

lede.off() #1

ledf.on() #0

ledg.off() #1

if numero == 3:

leda.off() #1

ledb.off() #1

ledc.off() #1

ledd.off() #1

lede.on() #0

ledf.on() #0

ledg.off() #1

if numero == 4:

leda.on()

ledb.off()

ledc.off()

ledd.on()

lede.on()

ledf.off()

ledg.off()

if numero == 5:

leda.off()

ledb.on()

ledc.off()

ledd.off()

lede.on()

ledf.off()

ledg.off()

if numero == 6:

leda.off()

ledb.on()

ledc.off()

ledd.off()

lede.off()

ledf.off()

ledg.off()

if numero == 7:

leda.off()

ledb.off()

ledc.off()

ledd.on()

lede.on()

ledf.on()

ledg.on()

if numero == 8:

leda.off()

ledb.off()

ledc.off()

ledd.off()

lede.off()

ledf.off()

ledg.off()

if numero == 9:

leda.off()

ledb.off()

ledc.off()

ledd.off()

lede.on()

ledf.off()

ledg.off()

time.sleep(0.1)

**Experimento 6.** Trabajar el concepto de PWM, para variar la iluminación de un LED RGB

**Objetivo**. Hacer que el LED RGB, emita diferentes colores utilizando el concepto PWM.

**Procedimiento**. Correr el programa siguiente.

LED\_RGB

from machine import Pin, PWM

import time

# Define los pines GPIO para el LED RGB

RED\_PIN = 27

GREEN\_PIN = 26

BLUE\_PIN = 25

# Define los canales PWM

red = PWM(Pin(RED\_PIN))

green = PWM(Pin(GREEN\_PIN))

blue = PWM(Pin(BLUE\_PIN))

# Define la frecuencia PWM

red.freq(1000)

green.freq(1000)

blue.freq(1000)

def set\_color(r, g, b):

red.duty(r)

green.duty(g)

blue.duty(b)

while True:

# Define los colores y espéra un momento

set\_color(1023, 0, 0) # Red

time.sleep(1)

set\_color(0, 1023, 0) # Green

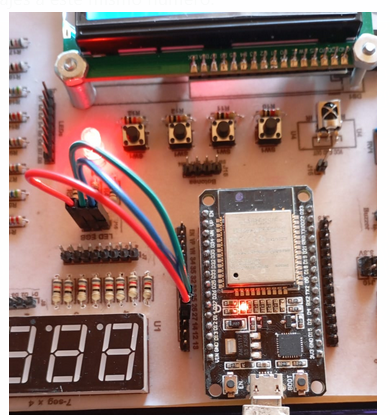
time.sleep(1)

set\_color(0, 0, 1023) # Blue

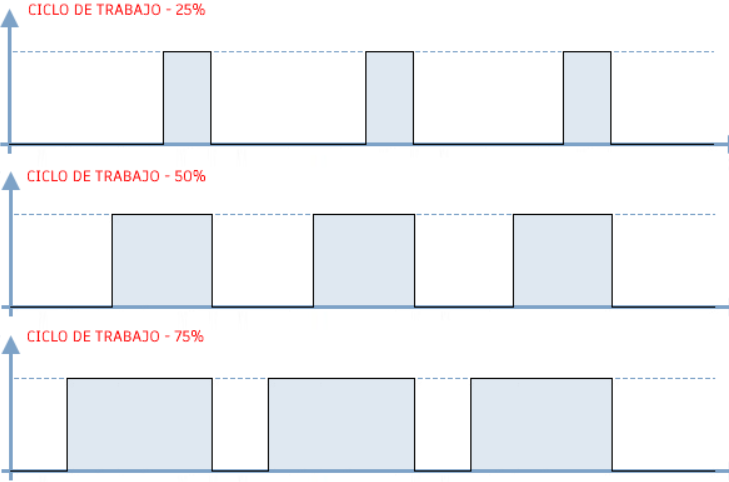
time.sleep(1)

set\_color(1023, 0, 1023) # purple

time.sleep(1)



En este caso de la librería machine, se importa, además del módulo PIN, el módulo PWM. Este módulo permite modificar la cantidad de energía que se puede enviar a través de un pin con valor lógico 1. El ciclo de trabajo, Duty Cycle, hace referencia a la cantidad real de energía que se envía por ese pin. O al porcentaje de tiempo en que el ciclo esta en nivel lógico 1.



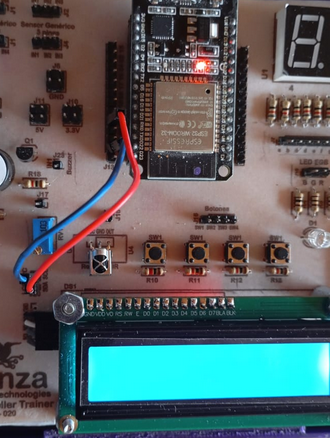
La frecuencia de funcionamiento del PWM está entre 1 y 1000 Hz.

Incluye una función, la función def set\_color(r, g, b):. Esta estructura ejecuta el bloque de instrucciones que tiene cada vez que se llama. En este caso le está dando un ciclo de PWM a cada uno de los colores R, G y B.

Describa que sucede con el LED RGB.

**Experimento 7.** Utilizar la pantalla 1602

**Objetivo**. Aprender a utilizar la pantalla LCD1602.



**Procedimiento.** Correr el siguiente código

LCD\_1602

from lcd1602 import LCD

import time

# Crea una instancia de la clase LCD y le asigna la variable lcd

lcd = LCD()

# Define el string " QUYNZA\n"

string = " QUYNZA\n"

# Presenta el string en la pantalla

lcd.message(string)

time.sleep(2)

# Define el string " Tecnologia!""

string = " Tecnologia!"

# Presenta el string en la pantalla

lcd.message(string)

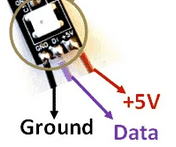
time.sleep(2)

# Clear the LCD screen

lcd.clear()

Aquí se tiene otra librería lcd1602 que incluye módulos especializados para utilizar la pantalla LCD 1602. Escriba otros mensajes.

**Experimento 8.** Utilizar la Cinta LED RGB



**Objetivo**. Aprender a utilizar la Cinta LED RGB.

**Procedimiento.** Correr los siguientes códigos

**1.** Cinta\_RGB1

from machine import Pin

import neopixel

import time

data\_pin = Pin(14, Pin.OUT)

num\_pixels = 8

pixels = neopixel.NeoPixel(data\_pin, num\_pixels)

# crea un controlador NeoPixel en el pin 14 para 8 pixeles

pixels[0] = [64,154,227]

pixels[1] = [128,0,128]

pixels[2] = [50,150,50]

pixels[3] = [255,30,30]

pixels[4] = [0,128,255]

pixels[5] = [99,199,0]

pixels[6] = [128,128,128]

pixels[7] = [255,100,0]

while True:

for i in range(num\_pixels):

# Actualiza la cinta RGB LED

pixels.write()

time.sleep\_ms(100)

Aquí se tiene otra librería neopixel que incluye módulos especializados para utilizar la cinta LED RGB. La función NeoPixel devuelve una lista pixels de 8 posiciones, para cada conjunto de 8 LEDs de la lista.

**2.** Cinta\_RGB2

from machine import Pin

import neopixel

import time

import random

# Define el numero de pixels para desplazar la luz

num\_pixels = 8

# Define el pin de datos pin para la cinta RGB LED

data\_pin = Pin(14, Pin.OUT)

# Define el objeto RGB LED strip

pixels = neopixel.NeoPixel(data\_pin, num\_pixels)

# El ciclo continuamente desplaza la luz

while True:

for i in range(num\_pixels):

# Genera un color aleatorio para el pixel corriente

color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))

# Cambia el pixel corriente con el color aleatorio

pixels[i] = color

# Actualiza la cinta RGB LED

pixels.write()

# Apaga el pixel corriente

pixels[i] = (0, 0, 0)

#Espera para controlar la velocidad de desplazamiento de la luz

time.sleep\_ms(100)

Aquí se tiene una nueva librería: random, que nos permite generar diferentes tipos de números aleatorios, entre diferentes rangos. Es muy útil pues se utiliza en muchas aplicaciones.

**Experimento 8.** Utilizar el zumbador

**Objetivo**. Aprender a utilizar el zumbador.

**Procedimiento.** Correr los siguientes códigos

**1.** Zumba1

import machine

import time

# Crea un objeto Pin que representa el pin 19 y lo define en modo salida

buzzer = machine.Pin(19, machine.Pin.OUT)

# Entra en un ciclo infinito

while True:

# Itera sobre los valores 0 to 3 usando un ciclo **for**

for i in range(4):

# Enciende el zumbador colocando su valor en 1.

buzzer.value(1)

# Pausa durante 0.2 segundos.

time.sleep(0.2)

# Apaga el zumbador

buzzer.value(0)

# Pausa durante 0.2 segundos.

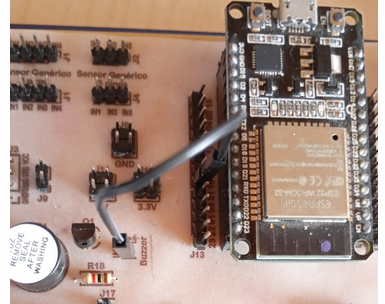
time.sleep(0.2)

# Pausa durante 1 segundos, antes de reiniciar el ciclo **while**

time.sleep(1)

En este código se incluye la estructura **for**, con la función **range**. Lo que hace es recorrer el bloque de instrucciones del ciclo **for** 4 veces.

**Experimento 9.** Utilizar el zumbador



**Objetivo**. Aprender a utilizar el zumbador.

**Procedimiento.** Correr los siguientes códigos

import machine

import time

# Define a tone function that takes as input a Pin object representing the buzzer, a frequency in Hz, and a duration in milliseconds

def tone(pin, frequency, duration):

pin.freq(frequency) # Set the frequency

pin.duty(512) # Set the duty cycle

time.sleep\_ms(duration) # Pause for the duration in milliseconds

pin.duty(0) # Set the duty cycle to 0 to stop the tone

#def rutine\_buzzer2(BUZZER\_PIN):

# Define the GPIO pin that is connected to the buzzer

buzzer = machine.PWM(machine.Pin(19))

# Define the frequencies of the notes in Hz

C5 = 523

D5 = 587

E5 = 659

F5 = 698

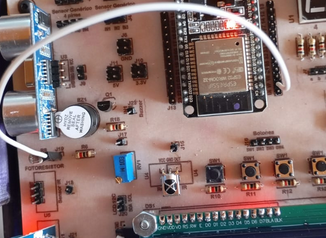
G5 = 784

A5 = 880

**Experimentos con sensores o dispositivos de salida**

Los siguientes experimentos utilizan dispositivos que implícitamente producen un valor numérico que el micro-controlador va a leer y va a mostrar sus resultados o va a producir algún efecto como los pulsadores. Aquí tenemos los siguientes dispositivos: pulsadores, la foto-resistencia,, la resistencia variable o reóstato, el sensor de rayos infra-rojos, el sensor de humedad y temperatura, receptor IR y el sensor de ultrasonido. También hay 3 pines disponibles para leer otro tipo de sensores que se ajusten a estas señales. La salida se da por la consola del IDE Thonny.

**Experimento 10.** La fotorresistencia



**Objetivo**. Aprender a utilizar la fotorresistencia.

**Procedimiento.** Correr los siguientes códigos. El resultado se lee por la consola.

from machine import ADC,Pin

import time

# Crea un objeto ADC que actúa sobre el pin

foto\_resistencia = ADC(Pin(36, Pin.IN))

# Configura la atenuación ADC a 11dB para un rango completo

foto\_resistencia.atten(foto\_resistencia.ATTN\_11DB)

while True:

# Lee un valor análogo bruto entre 0-4095

valor = foto\_resistencia.read()

# A mayor iluminación, menor resistencia

print(valor)

time.sleep(0.05)

#Ahora, lee el valor de resistencia y lo envía a la pantalla

from machine import ADC,Pin

import time

from lcd1602 import LCD

foto\_resistencia = ADC(Pin(36, Pin.IN))

# Configura la atenuación ADC a 11dB para un rango completo

foto\_resistencia.atten(foto\_resistencia.ATTN\_11DB)

value = foto\_resistencia.read()

print(value)

string = str(value)

time.sleep(0.05)

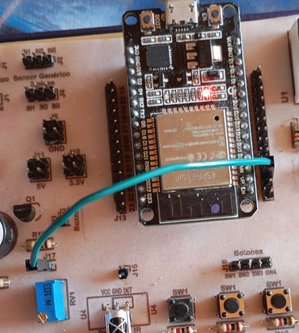
lcd = LCD()

# Presenta el string en la pantalla

lcd.message(string)

**Experimento 9.** El potenciómetro

**Objetivo**. Aprender a utilizar el potenciómetro



**Procedimiento.** Correr los siguientes códigos. Ahora el resultado se lee por la consola y la pantalla.

from machine import ADC, Pin, PWM

import time

pot = ADC(Pin(39, Pin.IN)) # Crea un objeto ADC que actúa sobre el pin

# Configura la atenuación ADC a 11dB para un rango completo

pot.atten(pot.ATTN\_11DB)

while True:

# Lee un valor análogo bruto entre 0-4095

valor = pot.read()

# Lee el voltaje in microvolts, lo convierte a voltios

voltaje = pot.read()/1000000

# Print the raw value and the voltage

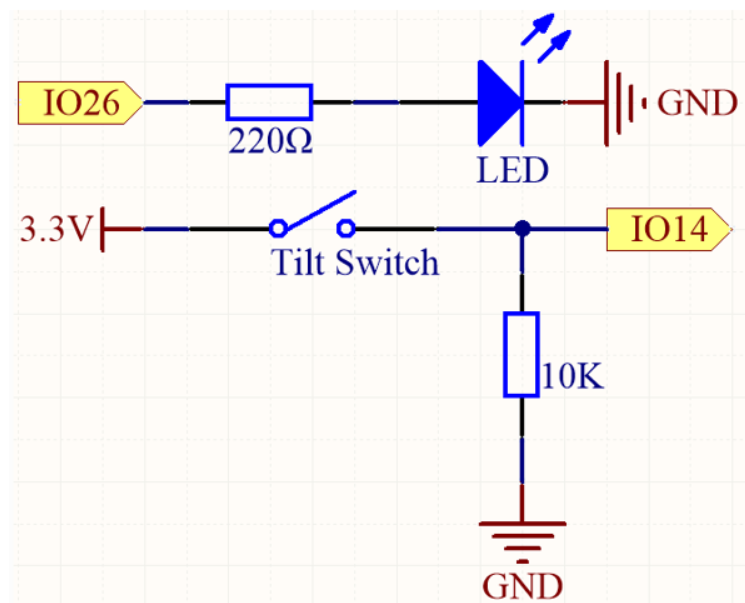
print(f"valor: {valor}, Voltaje: {voltaje}V")

# ESpera 0.5 segundos antes de hacer la nueva lectura

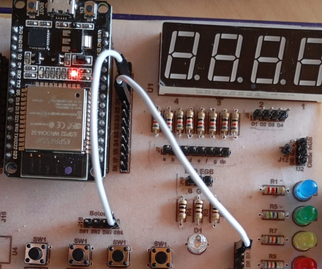
time.sleep(0.5)

**Experimento 10.** Utilización de un pulsador

**Objetivo**. Lee el presionar el pulsador



**Procedimiento.** Correr los siguientes códigos.



import machine

import time

boton = machine.Pin(14, machine.Pin.IN) # Button pin

led = machine.Pin(26, machine.Pin.OUT) # LED pin

while True:

# Al presionar se lee su valor

if boton.value() == 1:

# Se ilumina el LED

led.value(1)

time.sleep(0.5)

else:

# Se apaga el LED

led.value(0)

**Experimento 10.** Sensor de medida de distancia

**Objetivo**. Aprender a utilizar el sensor de medida de distancia

**Procedimiento.** Correr los siguientes códigos. Ahora el resultado se lee por la consola y la pantalla.

**Sensor de distancia HC-SR04**

****

El HC-SR04 es un sensor de distancias por ultrasonidos capaz de detectar objetos y calcular la distancia a la que se encuentra en un rango de 2 a 450 cm. El sensor funciona por ultrasonidos y contiene toda la electrónica encargada de hacer la medición. De muy pequeño tamaño, el HC-SR04 se destaca por su bajo consumo, gran precisión y bajo precio. Genera pulsos de alta frecuencia por el Trigger (derecho), que rebotan en los objetos y se reflejan hacia el sensor que los lee el ECHO (izquierdo).

**Características:**

* Dimensiones del circuito: 43 x 20 x 17 mm
* Tensión de alimentación: 5 Vcc
* Frecuencia de trabajo: 40 KHz
* Rango máximo: 4.5 m
* Rango mínimo: 1.7 cm
* Duración mínima del pulso de disparo (nivel TTL): 10 μS.
* Duración del pulso eco de salida (nivel TTL): 100-25000 μS.
* Tiempo mínimo de espera entre una medida y el inicio de otra 20 mS.

import machine

import time

# Define los pines trigger y echo para sensar la distancia

TRIG = machine.Pin(16, machine.Pin.OUT)

ECHO = machine.Pin(4, machine.Pin.IN)

# Calcula la distancia usando el sensor ultrasaonico

def distancia():

# Asegura que inicialmente el trigger este off

TRIG.off()

time.sleep\_us(2) # Espera durane 2 microsegundos

# Envia un pulso de 10-microsegundos al pin trigger

TRIG.on()

time.sleep\_us(10)

TRIG.off()

# Espera un momento para que el pin echo pase a high

while not ECHO.value():

pass

# Registra el tiempo cuando el pin echo va a high

time1 = time.ticks\_us()

# Espera a que el pin echo vaya a low

while ECHO.value():

pass

# Registra el tiempo cuando el pin echo va a low

time2 = time.ticks\_us()

# Calcula la diferencia de tiempo entre el registro de los dos tiempos

during = time.ticks\_diff(time2, time1)

# Calculate and return the distance (in cm) using the speed of sound (340 m/s)

return during \* 340 / 2 / 10000

# Mide continuamente la distancia y la imprime

while True:

dis = distancia()

print('Distancia: %.2f' % dis)

**Experimento 11.** Sensor de temperatura y humedad

**Objetivo**. Aprender a utilizar el sensor de temperatura y humedad

**Procedimiento.** Correr los siguientes códigos. Ahora el resultado se lee por la consola y la pantalla.

**SENSOR DE HUMEDAD Y TEMPERATURA DHT11**

Este sensor se caracteriza por tener la señal digital calibrada por lo que asegura una alta calidad y una fiabilidad a lo largo del tiempo, ya que contiene un conversor de 16 bits integrado, está constituido por dos sensores resistivos (NTC y humedad).

Tiene una excelente calidad y una respuesta rápida en las medidas, puede medir la humedad entre el rango 20% – aprox. 95% y la temperatura entre el rango 0ºC – 50ºC.

Cada sensor DHT11 está estrictamente calibrado en laboratorio, presentando una extrema precisión en la calibración. Los coeficientes de calibración se almacenan como programas en la memoria OTP, que son empleados por el proceso de detección de señal interna del sensor.

El protocolo de comunicación es a través de un único hilo (protocolo 1-wire), por lo tanto hace que la integración de este sensor en nuestros proyectos sea rápida y sencilla. Además presenta un tamaño reducido, un bajo consumo y la capacidad de transmitir la señal hasta 20 metros de distancia.

**ESPECIFICACIONES**

humedad relativa

Resolución: 16 bit

Repetibilidad: ± 1% HR

Precisión: En 25 ℃ ± 5% de humedad relativa

Intercambiabilidad: totalmente intercambiables

Tiempo de respuesta: 1 / e (63%) de 25 ℃ 6s

1m / s 6s aire

Histéresis: <± 0,3% HR

Estabilidad a largo plazo: <± 0.5% RH / yr in

temperatura

Resolución: 16 bit

Repetibilidad: ± 0,2 ℃

Rango: A los 25 ℃ ± 2 ℃

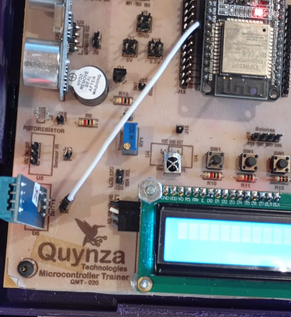
Tiempo de respuesta: 1 / e (63%) 10S

Características eléctricas

Fuente de alimentación: CC 3.5 ~ 5.5V

Corriente de alimentación: medida 0.3mA espera 60μ A

Periodo de muestreo: más de 2 segundos



import dht

import machine

import time

# Inicia el sensor DHT11

sensor = dht.DHT11(machine.Pin(5))

# Ciclo para medir continuamente la temperatura y la humedad

while True:

result = sensor.measure()

temp = sensor.temperature()

hume = sensor.humidity()

print("Temperature: ", temp)

print("Humidity: ", hume)

time.sleep(1)

Presentar humedad y temperatura por la pantalla LCD

import dht

import machine

import time

from lcd1602 import LCD

# Inicia el sensor DHT11 sensor y lo conecta al pin 5

sensor = dht.DHT11(machine.Pin(5))

# Inicia la pantalla LCD1602

lcd = LCD()

# Ciclo permanente para medir continuamente la temperatura y la humedad

while True:

try:

# Mide la temperatura y la humedad

sensor.measure()

# Toma los valores de temperatura y humedad

temp = sensor.temperature()

hume = sensor.humidity()

# Saca por consola los valores de temperatura y humedad

print("Temperatura: {}, Humedad: {}".format(temp, hume))

# Limpia la pantalla LCD

lcd.clear()

# Presenta la temperatura y la humedad en la pantalla LCD1602

lcd.write(0, 0, "Temp: {}\xDFC".format(temp))

lcd.write(0, 1, "Hume: {}%".format(hume))

# Wait for 2 seconds before measuring again

time.sleep(2)

except Exception as e:

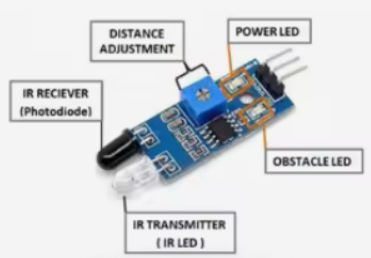
print("Error: ", e)

time.sleep(2)

**Experimento 12.** Sensor IR Infrared Obstacle Avoidance

**Objetivo**. Aprender a utilizar el módulo de evitar obstaculos

**Procedimiento.** Correr los siguientes códigos.



import machine

import time

ir\_avoid = machine.Pin(17, machine.Pin.IN, machine.Pin.PULL\_UP)

# Pin del módulo evitar

while True:

# Muestra los valores del módulo de evitar obstáculos

print(ir\_avoid.value())

time.sleep(0.1)

The sensor module adaptable to ambient light, having a pair of infrared emitting and receiving LEDs, transmitting LED emit infrared certain frequency, when the direction of an obstacle is detected (reflection surface), the infrared reflected is received by the reception LED, After a comparator circuit processing, the obstacle light is on, but the signal output interface output digital signal (a low-level signal), you can adjust the detection distance trim potentiometer, the effective distance range of 2 ~ 20cm, the working voltage of 3.3V- 5V. Detection range of the sensor can be obtained by adjusting potentiometer, with little interference, easy to assemble, easy to use features, can be widely used in robot obstacle avoidance, avoidance car, line count, black and white line tracking and many other applications.  
  
When the module detects an obstacle in front of the signal, the obstacle indicator lights on the board level, while the OUT port sustained low signal output, the module detects the distance 2 ~ 20cm, detection angle 35 °, the distance can detect potential is adjusted clockwise adjustment potentiometer, detects the distance increases; counterclockwise adjustment potentiometer, reducing detection distance.  
  
The sensor active infrared reflection detection, target reflectivity and therefore the shape is critical detection distance. Where the minimum detection distance black, white, maximum; small objects away from a small area, a large area from the Grand, the sensor module output port OUT port can be directly connected to the microcontroller IO can also be directly drive a 5V relay; Connection: VCC-VCC; GND-GND; OUT-IO,comparators LM393, stable, the module can be 3-5V DC power supply. When the power is turned on, the red power indicator lights, with the screw holes 3mm, easy fixed installation, board size: 3.2CM \* 1.4CM.

**Experimento 13.** Sensor IR Infrared Obstacle Avoidance

**Objetivo**. Aprender a utilizar el módulo de evitar obstaculos

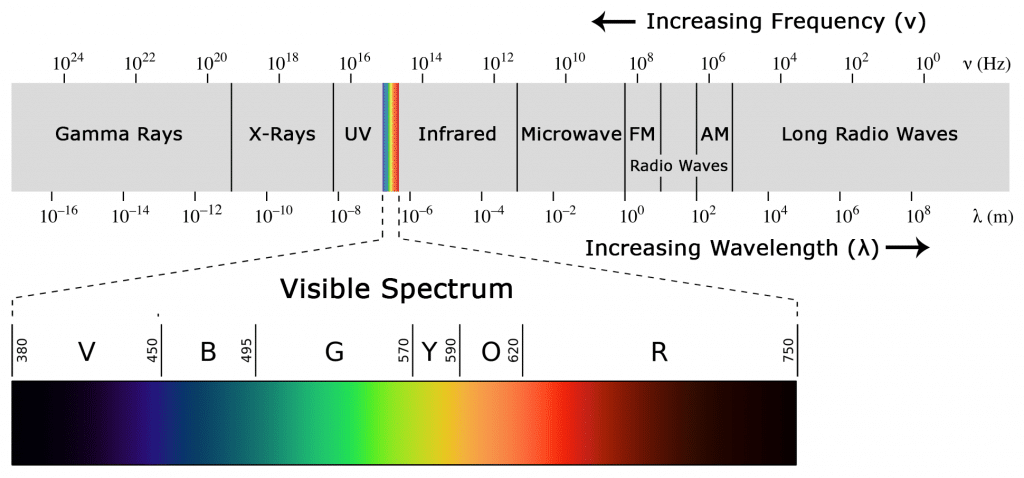
**Procedimiento.** Correr los siguientes códigos.

The [3-in-1 Smart Car and IOT Learning Kit](https://www.sunfounder.com/products/sunfounder-3-in-1-iot-smart-car-learning-ultimate-starter-kit?_pos=1&_sid=3dc627d5e&_ss=r&ref=5RPAW01H) from SunFounder has everything you need to learn how to master the Arduino. It includes all of the parts, wiring diagrams, code, and step-by-step instructions for 58 different robotics and internet of things projects that are super fun to build!

Now let’s get into the details…

## What is Infrared?

Infrared radiation is a form of light similar to the light we see all around us. The only difference between IR light and visible light is the frequency and wavelength. Infrared radiation lies outside the range of visible light, so humans can’t see it:

[](https://www.circuitbasics.com/wp-content/uploads/2017/05/Arduino-IR-Tutorial-Electromagnetic-Spectrum-Diagram.png)

Because IR is a type of light, IR communication requires a direct line of sight from the receiver to the transmitter. It can’t transmit through walls or other materials like WiFi or Bluetooth.

import time

from machine import Pin, freq

from ir\_rx.print\_error import print\_error

from ir\_rx.nec import NEC\_8

pin\_ir = Pin(15, Pin.IN) # IR receiver

# Decode the received data and return the corresponding key name

def decodeKeyValue(data):

if data == 0x16:

return "0"

if data == 0x0C:

return "1"

if data == 0x18:

return "2"

if data == 0x5E:

return "3"

if data == 0x08:

return "4"

if data == 0x1C:

return "5"

if data == 0x5A:

return "6"

if data == 0x42:

return "7"

if data == 0x52:

return "8"

if data == 0x4A:

return "9"

if data == 0x09:

return "+"

if data == 0x15:

return "-"

if data == 0x7:

return "EQ"

if data == 0x0D:

return "U/SD"

if data == 0x19:

return "CYCLE"

if data == 0x44:

return "PLAY/PAUSE"

if data == 0x43:

return "FORWARD"

if data == 0x40:

return "BACKWARD"

if data == 0x45:

return "POWER"

if data == 0x47:

return "MUTE"

if data == 0x46:

return "MODE"

return "ERROR"

# User callback

def callback(data, addr, ctrl):

if data < 0: # NEC protocol sends repeat codes.

pass

else:

print(decodeKeyValue(data))

ir = NEC\_8(pin\_ir, callback) # Instantiate the NEC\_8 receiver

# Show debug information

ir.error\_function(print\_error)

# keep the script running until interrupted by a keyboard interrupt (Ctrl+C)

try:

while True:

pass

except KeyboardInterrupt:

ir.close() # Close the receiver

**Experimento 14.** Utilización del zumbador

**Objetivo**. Aprender a utilizar el zumbador

**Procedimiento.** Correr los siguientes códigos.

import machine

import time

# Crea un objeto PIN que representa el pin 19 Pin y lo deja en modo salid

buzzer = machine.Pin(19, machine.Pin.OUT)

# Entra en un ciclo de 5 segs

time1 = time.ticks\_us()

during = 0

while during <= 5000000:

time2 = time.ticks\_us()

during = time.ticks\_diff(time2, time1)

# Itera con valores de 0 a 3 usando eñ ciclo for

for i in range(4):

# Pone el zumbador en 1

buzzer.value(1)

# Hace una pause durante 0.2 segs

time.sleep(0.2)

# Apaga el zumbador

buzzer.value(0)

# Hace una pause durante 0.2 segs

time.sleep(0.2)

# Hace una pause de 1 seg para volver a comenzar el ciclo for

time.sleep(1)Hace una pause durante 0.2 segs

time.sleep(0.2)

# Apaga el zumbador

buzzer.value(0)

# Hace una pause durante 0.2 segs

time.sleep(0.2)

# Hace una pause de 1 seg para volver a comenzar el ciclo for

time.sleep(1)

**Experimento 15.** Utilización del zumbador

**Objetivo**. Aprender a utilizar el zumbador

**Procedimiento.** Correr los siguientes códigos.

import machine

import time

# Crea un objeto PIN que representa el pin 19 Pin y lo deja en modo salida

buzzer = machine.Pin(19, machine.Pin.OUT)

# Define una funcion tono que toma como entrada un objeto Pin

# que representa el zumbador, una frecuencia in Hz, y una duración en milisegundos

def tono(pin, frequency, duration):

pin.freq(frequency) # Set the frequency

pin.duty(512) # Set the duty cycle

time.sleep\_ms(duration) # Pause for the duration in milliseconds

pin.duty(0) # Set the duty cycle to 0 to stop the tone

# Define el pin GPIO pin que se conecta al zumbador

buzzer = machine.PWM(machine.Pin(19))

# Define las frecuencias de las notas en Hz

C5 = 523

D5 = 587

E5 = 659

F5 = 698

G5 = 784

A5 = 880

B5 = 988

# Define las duratciones de las notas en milisegundos

quarter\_note = 250

half\_note = 300

whole\_note = 1000

# Define la melodia como una lista de tuples (nota, duración)

melody = [

(E5, quarter\_note),

(E5, quarter\_note),

(F5, quarter\_note),

(G5, half\_note),

(G5, quarter\_note),

(F5, quarter\_note),

(E5, quarter\_note),

(D5, half\_note),

(C5, quarter\_note),

(C5, quarter\_note),

(D5, quarter\_note),

(E5, half\_note),

(E5, quarter\_note),

(D5, quarter\_note),

(D5, half\_note),

(E5, quarter\_note),

(E5, quarter\_note),

(F5, quarter\_note),

(G5, half\_note),

(G5, quarter\_note),

(F5, quarter\_note),

(E5, quarter\_note),

(D5, half\_note),

(C5, quarter\_note),

(C5, quarter\_note),

(D5, quarter\_note),

(E5, half\_note),

(D5, quarter\_note),

(C5, quarter\_note),

(C5, half\_note)]

# Play the melody

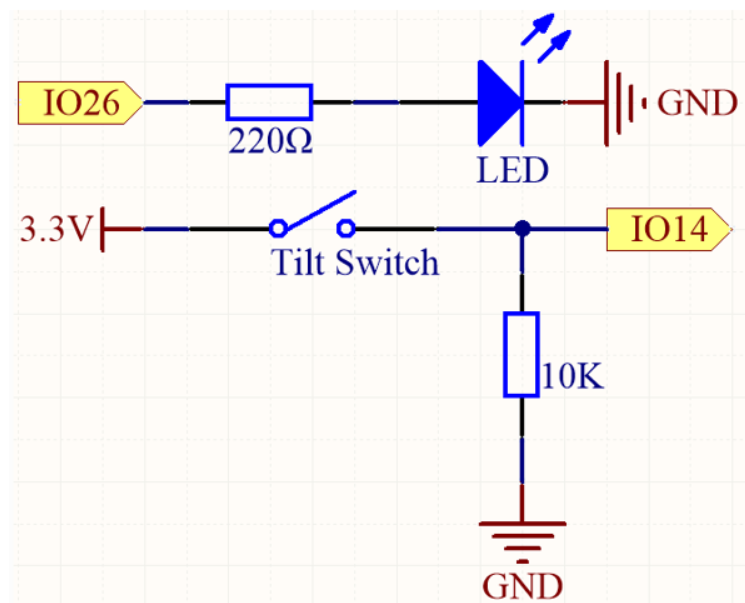
for note in melody:

tono(buzzer, note[0], note[1])

time.sleep\_ms(50)

**Experimento 16.** Utilización de un pulsador

**Objetivo**. Leer el presionar el pulsador



**Procedimiento.** Correr los siguientes códigos (no se ha corrfido).

import machine

import time

button = machine.Pin(14, machine.Pin.IN) # Button pin

led = machine.Pin(26, machine.Pin.OUT) # LED pin

while True:

# If the button is pressed by reading its value

if button.value() == 1:

# Turn on the LED by setting its value to 1

led.value(1)

# time.sleep(0.5)

else:

# Turn off the LED

led.value(0)

**Experimento 17.** Utilización de los pulsadores con el LED RGB

**Objetivo**. Usar diferentes composiciones de presión de los buttons y ver sus efectos en el LED RGB.

**Procedimiento.** Correr los siguientes códigos

import machine

import time

BUTTON1 = 27

BUTTON2 = 18

RGB\_R = 14

RGB\_G = 12

RGB\_B = 13

button1 = machine.Pin(BUTTON1, machine.Pin.IN) # Button pin

button2 = machine.Pin(BUTTON2, machine.Pin.IN) # Button pin

led\_R = machine.Pin(RGB\_R, machine.Pin.OUT) # LED pin

led\_G = machine.Pin(RGB\_G, machine.Pin.OUT) # LED pin

led\_B = machine.Pin(RGB\_B, machine.Pin.OUT) # LED pin

# Entra en un ciclo de 10 segs

time1 = time.ticks\_us()

during = 0

while during <= 10000000:

time2 = time.ticks\_us()

during = time.ticks\_diff(time2, time1)

# Si se presiona pulsador 1 y no se presiona pulsador 2 se ilumina el LED en rojo

if button1.value() and not button2.value():

# Turn on the LED by setting its value to 1

led\_R.value(1)

led\_G.value(0)

led\_B.value(0)

time.sleep(0.5)

# Si no se presiona pulsador 1 y se presiona el pulsador 2 se ilumina el LED en verde

elif not button1.value() and button2.value():

led\_R.value(0)

led\_G.value(1)

led\_B.value(0)

time.sleep(0.5)

elif button1.value() and button2.value():

# Si se presiona el pulsador 1 y se presiona el pulsador 2 a la vez se ilumina el LED en azul

led\_R.value(0)

led\_G.value(0)

led\_B.value(1)

time.sleep(0.5)

else:

# El LED deja de iluminar

led\_R.value(0)

led\_G.value(0)

led\_B.value(0)

time.sleep(0.5)

led\_R.value(0)

led\_G.value(0)

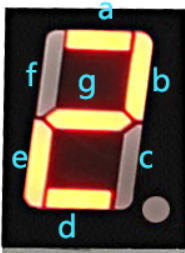
led\_B.value(0)

**Experimento 18.** Utilización del display de 7 segmentos

**Objetivo**. Con base en el método radint de la random, se toma un número aleatorio entre 0 y 9; y se muestra en el Display de 7 segmentos

Los LEDS de 7 segmentos, permiten, a través de la iluminación de ciertos segmentos mostrar los números decimales de 0 a 10. En el caso de la figura se tiene iluminados los segmentos a, b, g y d. que corresponden al binario 1011011 de la tabla.

josemiguel1886



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **a** | **b** | **c** | **d** | **e** | **f** | **g** | **Digito** |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 3 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 4 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 6 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 9 |

**Procedimiento** Con base en el método radint de la random, se toma un número aleatorio entre 0 y 9; y se muestra en el Display de 7 segmentos. Correr el siguiente código.

from machine import Pin

import time

import random

numero = random.randint(0, 9)

leda = Pin(16,Pin.OUT)

ledb = Pin(17,Pin.OUT)

ledc = Pin(18,Pin.OUT)

ledd = Pin(19,Pin.OUT)

lede = Pin(20,Pin.OUT)

ledf = Pin(21,Pin.OUT)

ledg = Pin(22,Pin.OUT)

while True:

if numero == 0:

leda.off() #1

ledb.off() #1

ledc.off() #1

ledd.off() #1

lede.off() #1

ledf.off() #1

ledg.on() #0

print("zero")

if numero == 1:

leda.on() #0

ledb.off() #1

ledc.off() #1

ledd.on() #0

lede.on() #0

ledf.on() #0

ledg.on() #0

if numero == 2:

leda.off() #1

ledb.off() #1

ledc.on() #0

ledd.off() #1

lede.off() #1

ledf.on() #0

ledg.off() #1

if numero == 3:

leda.off() #1

ledb.off() #1

ledc.off() #1

ledd.off() #1

lede.on() #0

ledf.on() #0

ledg.off() #1

if numero == 4:

leda.on()

ledb.off()

ledc.off()

ledd.on()

lede.on()

ledf.off()

ledg.off()

if numero == 5:

leda.off()

ledb.on()

ledc.off()

ledd.off()

lede.on()

ledf.off()

ledg.off()

if numero == 6:

leda.off()

ledb.on()

ledc.off()

ledd.off()

lede.off()

ledf.off()

ledg.off()

if numero == 7:

leda.off()

ledb.off()

ledc.off()

ledd.on()

lede.on()

ledf.on()

ledg.on()

if numero == 8:

leda.off()

ledb.off()

ledc.off()

ledd.off()

lede.off()

ledf.off()

ledg.off()

if numero == 9:

leda.off()

ledb.off()

ledc.off()

ledd.off()

lede.on()

ledf.off()

ledg.off()

time.sleep(0.1)

**Experimento 19.** Utilización del display de 7 segmentos

**Objetivo**. Con base en el método radint de la random, se toma un número aleatorio entre 0 y 9; y se muestra en el Display de 7 segmentos