

**UNIVERSIDAD TECNOLÓGICA DE**

**SAN LUIS RIO COLORADO**

**Investigación GPIO-PYTHON**

**MTRA. IRENE GARCIA**

**ALUMNO: VICTOR MANUEL GALVAN COVARRUBIAS**

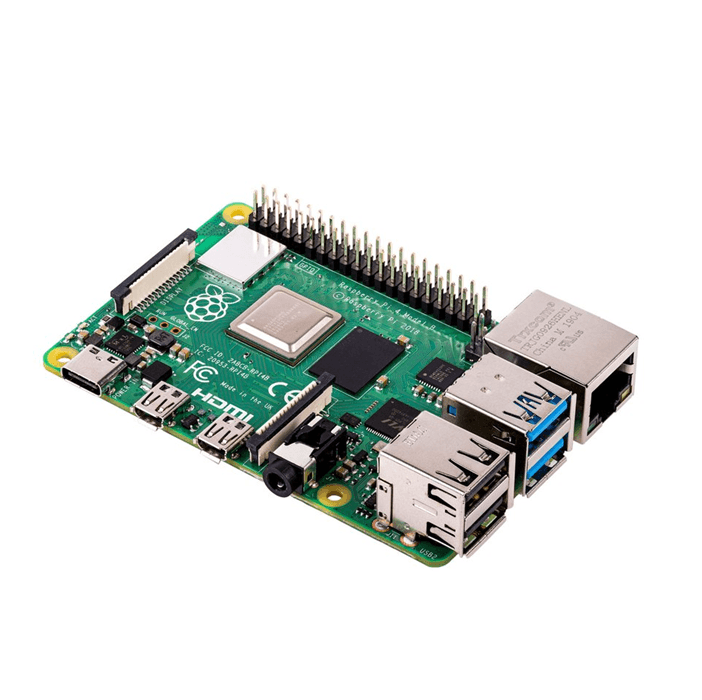
**TECNOLOGÍAS DE LA INFORMACIÓN**

**ÁREA DESARROLLO DE SOFTWARE MULTIPLATAFORMA**

San Luis Rio Colorado, Sonora Marzo, 2021

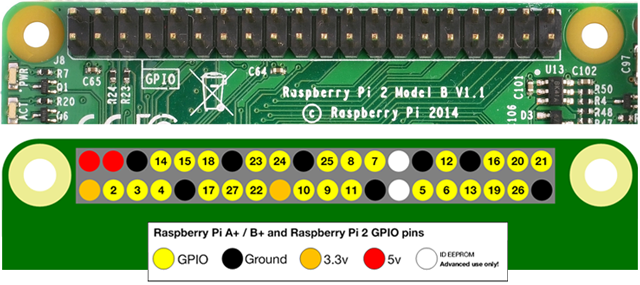
**1.- ¿Que es Raspberry?**

**Raspberry Pi**, es un ordenador de tamaño de tarjeta de crédito que se conecta a su televisor y un teclado. Es una placa que soporta varios componentes necesarios en un ordenador común. Es un pequeño ordenador capaz, que puede ser utilizado por muchas de las cosas que su PC de escritorio hace, como hojas de cálculo, procesadores de texto y juegos. También reproduce vídeo de alta definición.



**2.- ¿Que es GPIO?**

**General Purpose Input Output** (GPIO) es un sistema de entrada y salida de propósito general, es decir, consta de una serie de pines o conexiones que se pueden usar como entradas o salidas para múltiples usos. Estos pines están incluidos en todos los modelos de Raspberry Pi, aunque con diferencias.

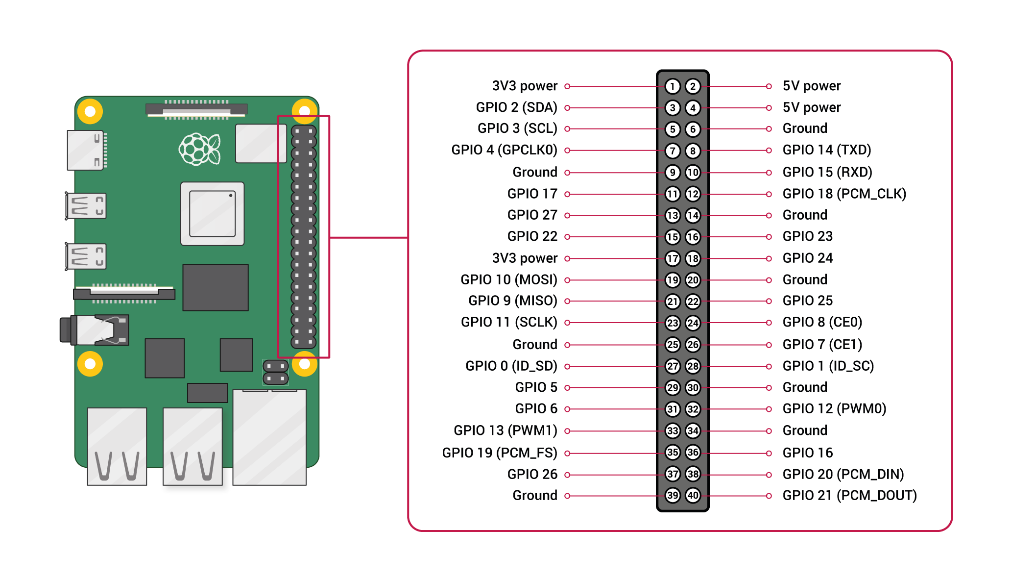


**3.- ¿Cómo acceder al puerto GPIO a través de Python?**

Para acceder con Python, es necesario importar las bibliotecas. Se necesitan dos bibliotecas: la biblioteca RPI.GPIO, para controlar los pines, y la biblioteca de tiempo, que permite a los usuarios crear sincronización entre cada transición del pin.

Luego, es necesario declarar el tipo de sistema de numeración. La opción BOARD usa los pines tal como están distribuidos en el Pi. Este conector y la numeración no cambian entre versiones. La opción BCM usa la numeración Broadcom SoC, que es diferente en distintas versiones de pines de Raspberry.

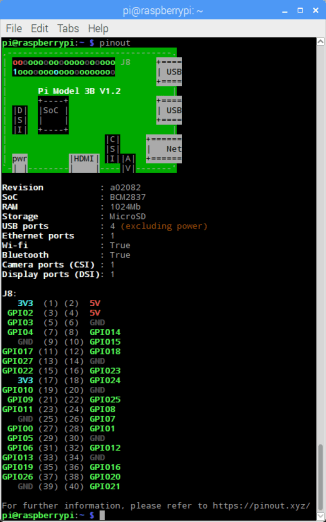
En nuestro programa, la función de comando GPIO.setmode usa y selecciona el número de la placa.



**4.-Tipos de conexiones de los pines del Raspberry.**

**BOARD GPIO**: este tipo de numeración de pines se refiere al número del pin en el enchufe, es decir, los números impresos en la placa, por ejemplo, P1. La ventaja de este tipo de numeración es que no cambiará, aunque cambie la versión de la placa.

**GPIO BCM**: la opción BCM se refiere al pin por “Broadcom SOC Channel”. Significan la designación del canal Broadcom SOC. El canal BCM cambia a medida que cambia el número de versión.



**5.- ¿Que es Python?**

Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.

**5.1.- Sintaxis Básica:**

>>> print("Hello, World!")

Hello, World!

if 5 > 2:

print("Five is greater than two!")

if 5 > 2:

print("Five is greater than two!")

## 5.2.- Identificadores Python:

## Un identificador comienza con una letra (de la A a la Z o de la a a la z) o con un guión bajo (\_) seguido de cero o más letras, guiones bajos y números.

## Python distingue mayúsculas de minúsculas.

## Python NO permite signos de puntuación como @, $ y %, excepto el guión bajo (\_).

## Se recomienda nombrar las clases comenzando por una letra mayúscula y el resto de los identificadores por una letra minúscula.

## Si un identificador comienza por un guión bajo, significa que es "privado".

## # Identificadores validos: radio RaDio contAlumnos Cont\_Alumnos num1

## # Identificadores NO validos: 2E2 Miércoles Cont-Alumnos Conjunto Primero print #palabras reservadas 1\_cantidad

## 5.3.- Palabras reservadas:

|  |  |  |
| --- | --- | --- |
| **False** | **Para indicar que algo es verdadero.** |  |
| **True** | **Para indicar que algo es falso.** |  |
| **None** | **Es utilizado para indicar la ausencia de un valor en un objeto determinado.** |  |
| **and / or** | **Representaciones lógicas de «y» y «ó», respectivamente.** | **>>> True and False**  **False**  **>>> True and True**  **True**  **>>> False and False**  **False**  **>>> True or False**  **True**  **>>> True or True**  **True**  **>>> False or False**  **False** |
| **as** | **Su función es dual. Puede utilizarse al momento de importar un determinado objeto de un módulo, para asignar un nombre diferente.** | **from keyword import kwlist as keyword\_list** |
| **assert** | **Durante la depuración (si \_\_debug\_\_ es True), assert permite especificar una expresión que lanzará AssertionError en caso de ser evaluada como falsa.** | **>>> assert 1 == 2**  **Traceback (most recent call last):**  **File "<stdin>", line 1, in <module>**  **AssertionError**  **>>> assert 1 == 1** |
| **async** | **Se emplea para definir una función como asincrónica.** | **async def f():**  **pass** |
| **await** | **Solo puede ser invocado desde una función asincrónica y sobre otra función de esta índole. Pausa la ejecución de aquella hasta que el resultado de esta esté disponible.** | **import asyncio**  **async def f():**  **await asyncio.sleep(1)** |
| **continue** | **Detener completamente un bucle (break) o detener únicamente la iteración actual y saltar a la siguiente (continúe).** |  |
| **break** |  |
| **class** | **Define una clase.** | **>>> class C:**  **... def \_\_init\_\_(self):**  **... print("Hello world!")**  **...**  **>>> c = C()**  **Hello world!** |
| **def** | **Define una función.** | **>>> def func(a):**  **... print(a)**  **...**  **>>> func("Hello world!")**  **Hello world!** |
| **del** | **En caso de utilizarse seguido de un objeto, elimina su referencia.** | **>>> a = 1**  **>>> a**  **1**  **>>> del a**  **>>> a**  **Traceback (most recent call last):**  **File "<stdin>", line 1, in <module>**  **NameError: name 'a' is not defined** |
| **En diccionarios, remueve el elemento del mismo. En listas, además, reordena el conjunto de elementos para llenar el espacio vacío.** | **>>> d = {"Yes": 1, "No": 2}**  **>>> d["Yes"]**  **1**  **>>> del d["Yes"]**  **>>> d["Yes"]**  **Traceback (most recent call last):**  **File "<stdin>", line 1, in <module>**  **KeyError: 'Yes'**  **>>> d**  **{'No': 2}** |
| **if / elif / else** | **Instrucciones permiten condicionar la ejecución de uno o varios bloques de sentencias al cumplimiento de una o varias condiciones.** | **if a:**  **# bool(a) es True**  **print("a")**  **elif b:**  **# bool(a) es False**  **# bool(b) es True**  **print("b")**  **elif c:**  **# bool(a) es False**  **# bool(b) es False**  **# bool(c) es True**  **print("c")**  **else:**  **# Todas las anteriores son falsas.**  **print("Ninguna de las anteriores.")** |
| **try / except / else / finally** | **Las cláusula try / except permite capturar una excepción dentro de una determinada porción de código.** | **try:**  **func()**  **except Exception:**  **print("Ha ocurrido un error.")** |
| **En caso de no haberse propagado ninguna excepción, el flujo del programa seguirá luego del término else, en caso de encontrarse, al finalizar la ejecución de func()** | **else:**  **print("Ejecutado exitosamente.")** |
| **Por último, el flujo será enviado luego del término finally al finalizar la ejecución del código anterior, independientemente si han surgido errores.** | **finally:**  **cleanup()** |
| **En caso de ocurrir un error en las cláusulas except o else, cleanup() será llamada antes de propagarse la excepción, para permitir realizar una limpieza, principal diferencia con el siguiente código.** | **try:**  **func()**  **except Exception:**  **print("Ha ocurrido un error.")**  **else:**  **print("Ejecutado exitosamente.")**  **cleanup()** |
| **for** | **Permite recorrer los elementos de un objeto iterable (listas, tuplas, diccionarios, etc.).** | **>>> for i in (1, 2, 3):**  **... print(i)**  **...**  **1**  **2**  **3** |
| **from** | **Importa uno o más objetos de un módulo, prescindiendo del nombre de éste como prefijo al ser utilizado.** | **>>> from keyword import kwlist, iskeyword**  **>>> iskeyword("assert")**  **True** |
| **global** | **Permite modificar el valor de un objeto global desde un ámbito con menor alcance (una función, por ejemplo).** | **>>> a = 1**  **>>> def f():**  **... global a**  **... a = 2**  **...**  **>>> f()**  **>>> a**  **2** |
| **import** | **Importa un módulo, o bien un objeto de éste si es utilizado junto a from.** | **>>> import keyword**  **>>> keyword.iskeyword("assert")**  **True** |
| **in** | **Determina la existencia de un determinado valor dentro de una lista, tupla, diccionario o cualquier objeto iterable.** | **>>> 2 in (1, 2, 3)**  **True**  **>>> "Hello" in "Hello, world!"**  **True**  **>>> "Si" in {"Yes": 1, "No": 2}**  **False** |
| **is** | **Determina si dos objetos son iguales, a diferencia de los signos de equivalencia (==), que determinan si ambos tienen el mismo valor.** | **>>> class Number:**  **... def \_\_init\_\_(self, number):**  **... self.number = number**  **... def \_\_eq\_\_(self, other):**  **... return self.number == other**  **...**  **>>> a = Number(1)**  **>>> b = Number(1)**  **>>> a == b**  **True**  **>>> a is b**  **False**  **>>> c = b**  **>>> b is c**  **True** |
| **lambda** | **Se trata de crear funciones de manera rápida, just in time, sobre la marcha, para prototipos ligeros que requieren únicamente de una pequeña operación o comprobación. Por lo tanto, toda función lambda también puede expresarse como una convencional (pero no viceversa).** | Función lambda |
| **nonlocal** | **Permite modificar el valor de un objeto que ha sido creado en un ámbito anterior.** | **>>> def a():**  **... i = 1**  **... def b():**  **... nonlocal i**  **... i = 2**  **... b()**  **... print(i)**  **...**  **>>> a()**  **2** |
| **not** | **Operador lógico «no».** | **>>> not True**  **False**  **>>> not False**  **True** |
| **pass** | **Esta palabra reservada carece de función alguna. Es utilizada para rellenar espacios requeridos por Python para evitar errores de sintaxis.** | **>>> def f():**  **... pass**  **...**  **>>> f()** |
| **raise** | **Lanza una excepción.** | **>>> raise NameError**  **Traceback (most recent call last):**  **File "<stdin>", line 1, in <module>**  **NameError** |
| **return** | **Dentro de una función, especifica el valor de retorno.** | **>>> def f():**  **... return 1**  **...**  **>>> f()**  **1** |
| **while** | **Ejecuta un bloque de código mientras que la expresión sea verdadera.** | **>>> i = 0**  **>>> while i < 2:**  **... print(i)**  **... i = i + 1**  **...**  **0**  **1** |
| **with** | **Permite encapsular la ejecución de un bloque de código, de modo que la inicialización y finalización de un objeto es realizada automáticamente por Python, utilizando las funciones \_\_enter\_\_ y \_\_exit\_\_.** | **with open("info.txt"):**  **raise Exception** |
| **yield** | **Se comporta al igual que return, con la diferencia que en lugar de retornar un único valor, retorna elementos que conforman un generador (un objeto iterable que puede recorrerse una vez, ya que el contenido no es almacenado en la memoria), por lo que puede emplearse múltiples veces en una misma función.** | **>>> def f():**  **... yield 1**  **... yield 2**  **... yield 3**  **...**  **>>> g = f()**  **>>> for i in g:**  **... print(i)**  **...**  **1**  **2**  **3**  **>>> for i in g:**  **... print(i)**  **...** |

## 5.4.- Estructura de las líneas:

## Un programa codificado con lenguaje Python está dividido en varias líneas lógicas y cada una de ellas finaliza con una nueva línea. Si la línea está en blanco, es ignorada por el intérprete de Python.

## Por otra parte, las órdenes y líneas se agrupan unas dentro de otras mediante sangrado o espaciado. Y para unir dos o más líneas, se emplea la barra inclinada hacia la izquierda.

## Otra posibilidad es que queramos realizar varias acciones o peticiones en una única línea. Para ello, podemos combinar todas las instancias empleando como separador el símbolo de punto y coma.

## 5.5.- Comentarios y citas en Python:

## En Python, los comentarios empiezan con el carácter hash (#). Nosotros podemos leernos, es más, deberíamos, ya que nos aportarán información sobre el resto de código, pero el intérprete lo ignorará.

## Los comentarios pueden incluirse en una línea propia o al lado de una orden. Siempre que tenga l símbolo indicado, Python lo interpretará como comentario y no como código válido.

## En cuanto a las citas, Python admite las comillas simples o dobles. El texto incluido dentro de las comillas será el que se mostrará en pantalla o el que se tendrá en cuenta para la orden programada. Si necesitamos citar un texto extenso, nos será de utilidad las triples comillas, que permiten citar textos que ocupan varias líneas.

## # Comentario de una línea

## ’’’

## Comentario de varias líneas

## ’’’