# Módulo de Programação Python

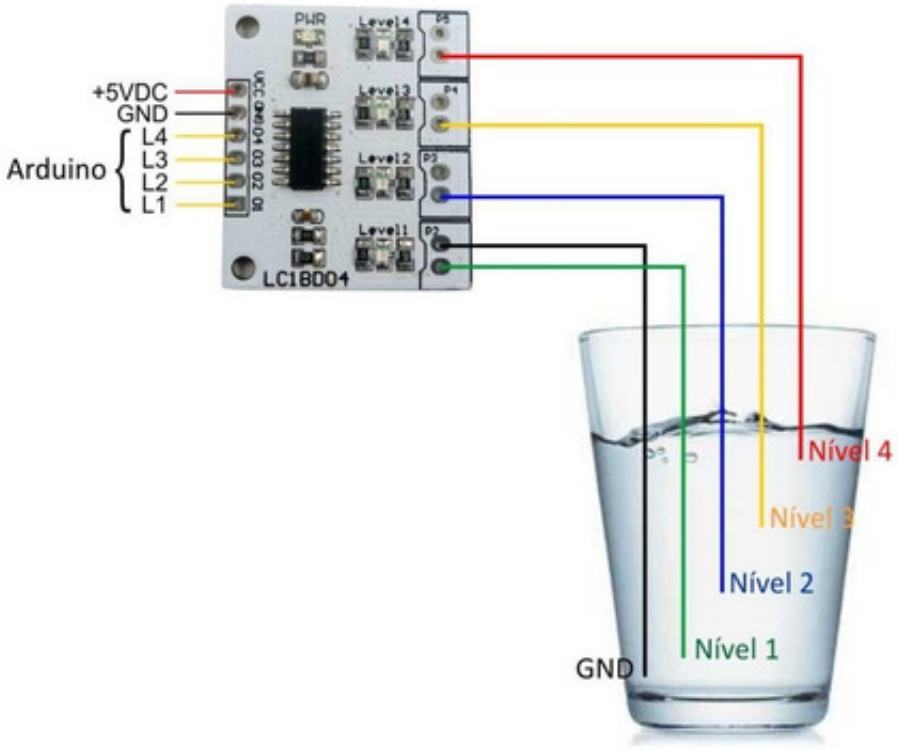
## Trilha Python - Aula 21: Modulos e Pacotes - Avançado


**Objetivo**: Apresentar como trabalhar para implementar pacotes e módulos em **Python** utilizando o **Poetry**.

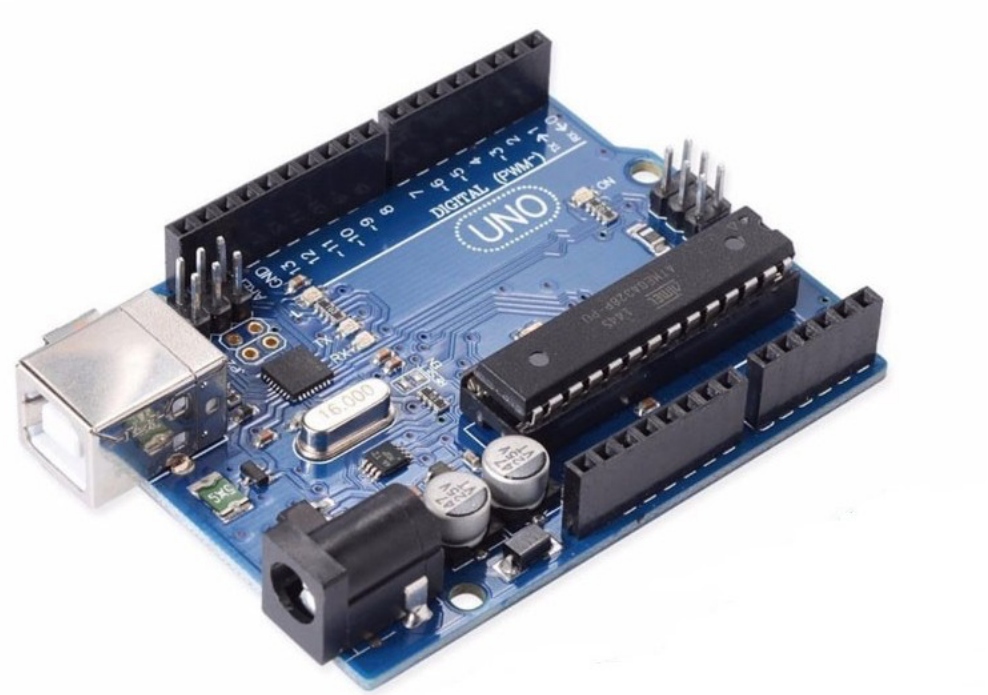
## Um problema para resolver


FORTLEV

# Sensor de Nível de Água




LED de força
PWR
+5VDC
GND
L4
L3
L2
L1
VCC GND 04 03 02 01
Level4
Level3
Level2
Level1
P5
P4
P3
P2
LC1BD04
LEDs indicadores de nível
4 níveis lógicos para:
- Arduino
- Microcontroladores
- ARM (STM32)

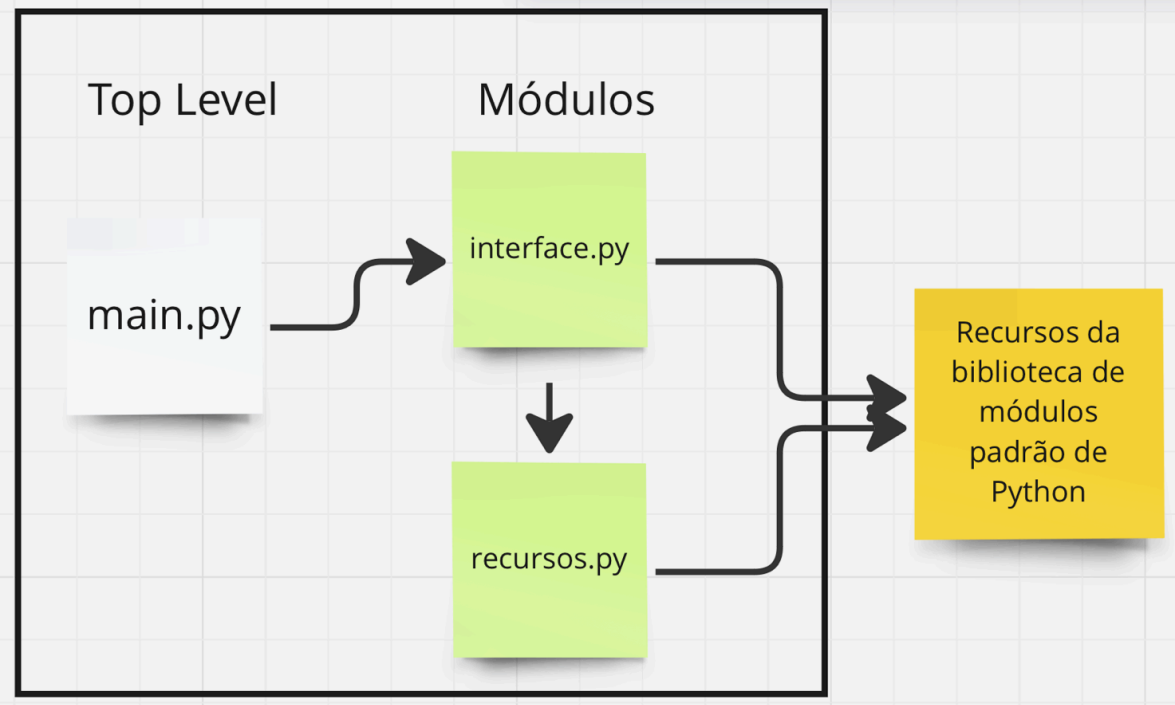

DIGITAL (PWM~)
UNO
ON
16.000

# Recapitulando o visto até aqui sobre módulos

## Módulos

Módulos em **Python** representam a unidade mais alta de organização de um programa, capas de armazenar códigos e dados para serem reutilizados. Os mecanismos disponíveis para carregar os recursos dos módulos foram pensados visando serem simples de usar e, na medida do possível, para minimizar conflitos de nomes.

- Um programa em Python consiste, basicamente, em um ou mais arquivos de texto contendo declarações Python.
- Um dos arquivo é o arquivo principal, que pode fazer uso ou não, de arquivos suplementares que são chamados de módulos. O arquivo principal passa a ser então o módulo `__main__` .



## Como funciona o processo de importar um módulo?

- Procura-se o arquivo do módulo;
- Uma vez encontrado o arquivo é compilado para *byte code*;
  - Explore a pasta `__pycache__` onde estão os módulos que você está usando;
- Se executa o módulo para gerar os objetos nele definidos;

Onde e em qual ordem buscar pelos arquivos de módulos

1. Na pasta da aplicação;
2. No caminho do **PYTHONPATH**;
3. No caminho padrão das bibliotecas;
4. Nas pastas indicadas nos arquivos .pth;
5. Nos sítios definidos por bibliotecas de terceiros;

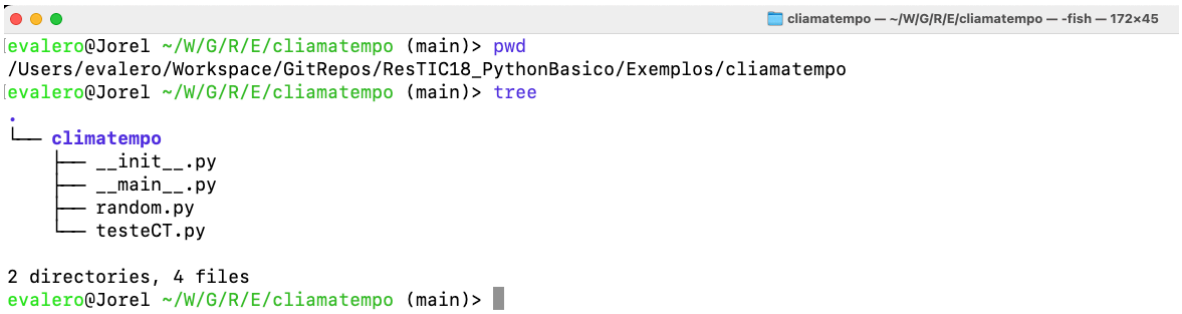
Podemos utilizar

a. Módulos da biblioteca padrão de **Pytho**;

b. Módulos desenvolvidos por terceiros;

c. Nossos próprios módulos.

Vejamos o seguinte exemplo.




`random.py`

```
1
2   import numpy as np
3
4   def random(size):
5       data = np.random.random(size)
6       return data
```

`__main__.py`

```
1  import random
2  #import sys
3
4  x = random.random(10)
5  print(type(x))
6
```

In [ ]:
```
# executando desde a pasta climatempo
# python climatempo/__main__.py
# ?????????
```

Vamos modificar o executável para saber o que está acontecendo.

```
evalero@Jorel ~/W/G/R/E/cliamatempo (main)> python -m climatempo
/Users/evalero/Workspace/GitRepos/ResTIC18_PythonBasico/Exemplos/cliamatempo
/opt/anaconda3/envs/pyTIC18/lib/python310.zip
/opt/anaconda3/envs/pyTIC18/lib/python3.10
/opt/anaconda3/envs/pyTIC18/lib/python3.10/lib-dynload
/opt/anaconda3/envs/pyTIC18/lib/python3.10/site-packages
evalero@Jorel ~/W/G/R/E/cliamatempo (main)>
```

```
evalero@Jorel ~/W/G/R/E/cliamatempo (main) [1]> python climatempo/__main__.py
/Users/evalero/Workspace/GitRepos/ResTIC18_PythonBasico/Exemplos/cliamatempo/climatempo
/opt/anaconda3/envs/pyTIC18/lib/python310.zip
/opt/anaconda3/envs/pyTIC18/lib/python3.10
/opt/anaconda3/envs/pyTIC18/lib/python3.10/lib-dynload
/opt/anaconda3/envs/pyTIC18/lib/python3.10/site-packages
evalero@Jorel ~/W/G/R/E/cliamatempo (main)>
```

**sys** — System-specific parameters and functions (https://docs.python.org/3/library/sys.html)

Este módulo fornece acesso a algumas variáveis utilizadas ou mantidas pelo interpretador e a funções que interagem fortemente com o interpretador.

## pipenv

O `pipenv` (https://pipenv.pypa.io/en/latest/) é uma ferramenta que combina `virtualenv` com `pip`. Ele permite criar ambientes virtuais e gerenciar pacotes Python para seus projetos. Ele foi projetado para ser mais fácil de utilizar do que usar `virtualenv e pip` separadamente.

Lembram de como p `pipenv` funciona?

Para utilizar esta ferramente, primeiramente é necessário criar uma pasta para o projeto e então ativar o ambiente virtual:

In [ ]:
```
# pipenv shell
# para sair do shell
# exit
```

Reparem que ele funciona um pouco diferente que as ferramentas anteriores. Agora é necessário sair do *shell* com o comando `exit` . Ele também cria uma pasta `Pipfile` para gerenciar os pacotes deste ambiente isolado. A instalação de pacotes também é feita de forma diferente:

In [ ]:
```
# pipenv install numpy
```

O comando anterior vai instalar o `numpy` e suas dependências na pasta `Pipfile` da pasta atual.

Vamos a voltar a falar do `pipenv` , quando falemos de gerenciamento de projetos e ambientes virtuais com `poetry` .

# Poetry

Segundo a documentação disponível no site do **Poetry** (https://python-poetry.org/docs/) "**Poetry** é uma ferramenta para gerenciamento de dependências e empacotamento em **Python**. Ele permite que você declare as bibliotecas das quais seu projeto depende e irá gerenciá-las (instalá-las/atualizá-las) para você. O **Poetry** fornece um arquivo de bloqueio para garantir instalações reprodutíveis e pode construir seu projeto para distribuição."

O **Poetry** traz uma serie de padrões de desenvolvimento de software em **Python**. Ele permite criar, de forma automática, toda a árvore de diretórios necessária para desenvolvimento padronizado de aplicações **Python**.

Vamos ver como ele funciona. Primeiramente vamos criar um ambiente virtual, por exemplo utilizando o conda.

In [1]:
```
# conda create --name poetryPy python=3.10
```

In [2]:
```
# conda activate poetryPy
```

In [3]:
```
# python -m pip install poetry
```

In [4]:
```
# pip list
# pip freeze > requirements.txt
```

Agora vamos criar um novo projeto com **Poetry**.

In [ ]:
```
#  poetry new climatempo
```

```
evalero@Jorel ~/W/G/R/E/exemPoetry (main)> poetry new climatempo
Created package climatempo in climatempo
evalero@Jorel ~/W/G/R/E/exemPoetry (main)> tree climatempo
climatempo
├── README.md
├── climatempo
│   └── __init__.py
├── pyproject.toml
└── tests
    └── __init__.py

3 directories, 4 files
evalero@Jorel ~/W/G/R/E/exemPoetry (main)>
```

Uma vez criado o projeto posso mudar para a pasta do projeto e instalar os pacotes com o comando `add` .

Opções

- --group (-G): O grupo ao qual adicionar a dependência.
- --dev (-D): Adiciona pacote como dependência de desenvolvimento. (Obsoleto, use -G dev)
- --editable (-e): Adicione dependências vcs/path como editáveis.
- --extras (-E): Extras para ativar para a dependência. (vários valores permitidos)
- --optional: Adicione como uma dependência opcional.
- --python: versão do Python para a qual a dependência deve ser instalada.
- --platform: Plataformas para as quais a dependência deve ser instalada.
- --source: Nome da fonte a ser usada para instalar o pacote.
- --allow-prereleases: aceita pré-lançamentos.
- --dry-run: gera as operações, mas não executa nada (habilita implicitamente –verbose).
- --lock: Não execute a instalação (apenas atualize o arquivo de bloqueio).

In [6]:
```
# poetry add numpy
#    Using version ^1.26.3 for numpy
# poetry add pandas
#    Using version ^2.1.4 for pandas
# poetry add seaborn
#    Using version ^0.13.1 for seaborn
```

Veja com está ficando o arquivo `.toml` ...

## Poetry pyproject.toml

O comando `new` cria automaticamente uma versão padrão deste arquivo.

A documentação do arquivo `pyproject.toml` está disponível aqui (https://python-poetry.org/docs/pyproject/) e dela podemos destacar:

### Seção `tool.poetry`

- `name` : Define o nome do pacote. Este campo é obrigatório e deve ser um nome válido (https://peps.python.org/pep-0508/#names).
- `version` : Define qual aversão atual do pacote. Este campo é obrigatório e deve ser um string de versão válida (https://peps.python.org/pep-0440/).
- `description` : Uma breve descrição do pacote. Este campo é obrigatório.
- `license` : A licença do pacote. A notação recomendada para as licenças mais comuns é (em ordem alfabética).
  - Apache-2.0
  - BSD-2-Clause
  - BSD-3-Clause
  - BSD-4-Clause
  - GPL-2.0-only
  - GPL-2.0-or-later
  - GPL-3.0-only
  - GPL-3.0-or-later
  - LGPL-2.1-only
  - LGPL-2.1-or-later
  - LGPL-3.0-only
  - LGPL-3.0-or-later
  - MIT Este campo é opcional, mas é altamente recomendável fornecê-lo. Mais identificadores estão listados no SPDX Open Source License Registry (https://spdx.org/licenses/).

Se o seu projeto for proprietário e não usar uma licença específica, você poderá definir esse valor como Proprietary.

- `authors` : Os autores do pacote. Os autores do pacote. Este campo é obrigatório e composto por uma lista de autores que deve conter conter pelo menos um autor. Os autores devem estar no formato nome .
- `maintainers` : Os mantenedores do pacote. Este é um campo opcional e é composto de uma lista de mantenedores que deve ser distinta dos autores.
- `scripts` : Aqui são descritos os scripts ou executáveis que serão instalados durante a instalação do pacote.