1. We have visibility modifiers that control when and where the function can be called from: private means it's only callable from other functions inside the contract; internal is like private but can also be called by contracts that inherit from this one; external can only be called outside the contract; and finally public can be called anywhere, both internally and externally.

Since Solidity knows two kinds of function calls (internal ones that do not create an actual EVM call (also called a “message call”) and external ones that do), there are four types of visibilities for functions and state variables.

Functions have to be specified as being external, public, internal or private. For state variables, external is not possible.

**external:**

External functions are part of the contract interface, which means they can be called from other contracts and via transactions. An external function f cannot be called internally (i.e. f() does not work, but this.f() works). External functions are sometimes more efficient when they receive large arrays of data.

**public:**

Public functions are part of the contract interface and can be either called internally or via messages. For public state variables, an automatic getter function (see below) is generated.

**internal:**

Those functions and state variables can only be accessed internally (i.e. from within the current contract or contracts deriving from it), without using this.

**private:**

Private functions and state variables are only visible for the contract they are defined in and not in derived contracts.

**Note**

Everything that is inside a contract is visible to all observers external to the blockchain. Making something private only prevents other contracts from accessing and modifying the information, but it will still be visible to the whole world outside of the blockchain.

The visibility specifier is given after the type for state variables and between parameter list and return parameter list for functions.

**pragma solidity** >=0.4.16 <0.6.0;

**contract** C {

**function** f(uint a) **private** **pure** **returns** (uint b) { **return** a + 1; }

**function** setData(uint a) **internal** { data = a; }

uint **public** data;

}

Functions can be defined inside and outside of contracts.

Functions outside of a contract, also called “free functions”, always have implicit internal [visibility](https://docs.soliditylang.org/en/v0.8.15/contracts.html#visibility-and-getters). Their code is included in all contracts that call them, similar to internal library functions.

*// SPDX-License-Identifier: GPL-3.0*

**pragma solidity** >=**0.7.1** <**0.9.0**;

function sum(uint[] memory arr) pure returns (uint s) {

for (uint i = 0; i < arr.length; i++)

s += arr[i];

}

**contract** **ArrayExample** {

bool found;

function f(uint[] memory arr) public {

*// This calls the free function internally.*

*// The compiler will add its code to the contract.*

uint s = sum(arr);

require(s >= 10);

found = true;

}

}

**Note**

Functions defined outside a contract are still always executed in the context of a contract. They still have access to the variable this, can call other contracts, send them Ether and destroy the contract that called them, among other things.

The main difference to functions defined inside a contract is that free functions do not have direct access to storage variables and functions