**Assignment 13**

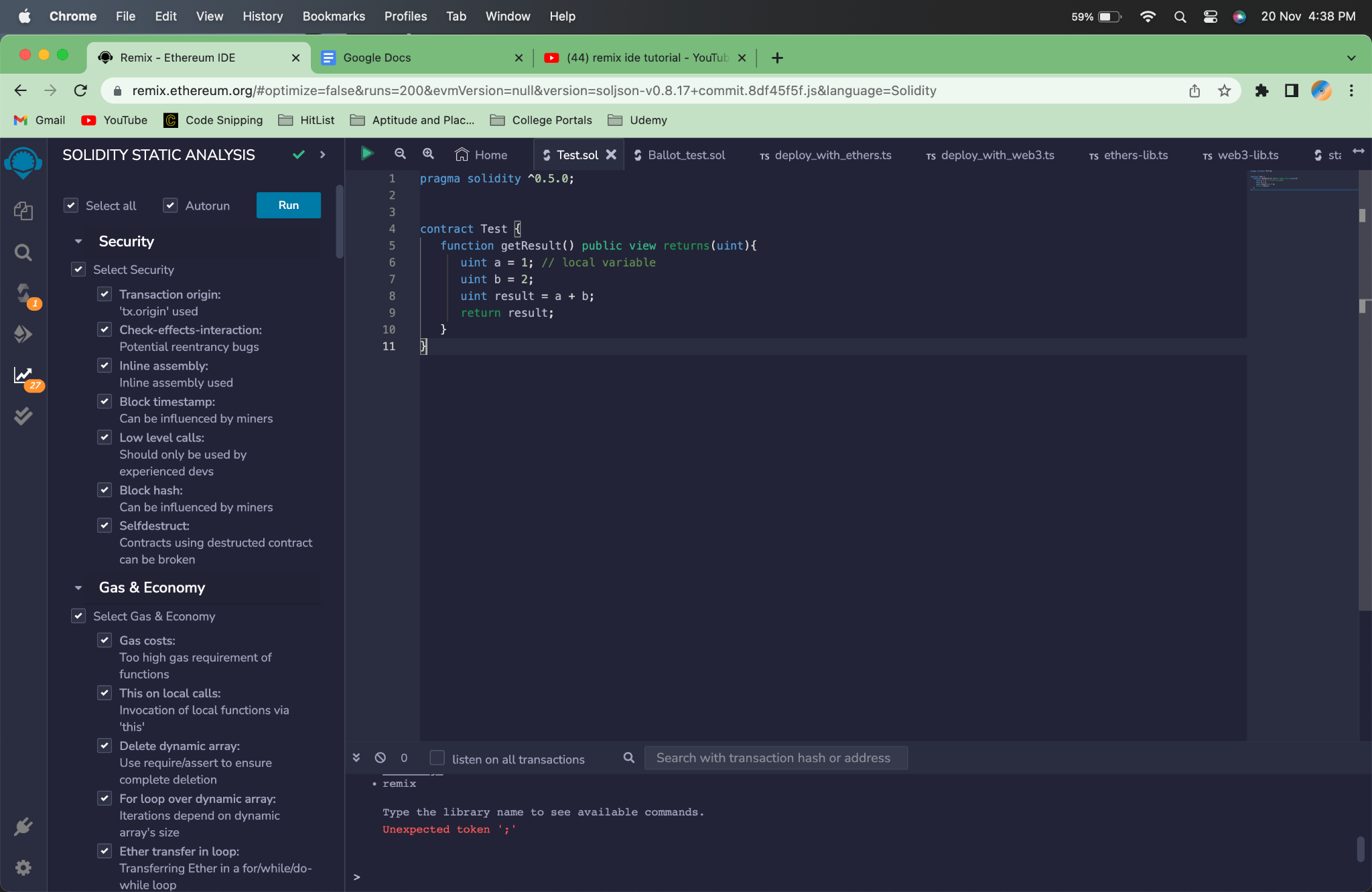
**Name**: Bhavin Patil

**Roll No:** 66

**TY-CS-D**

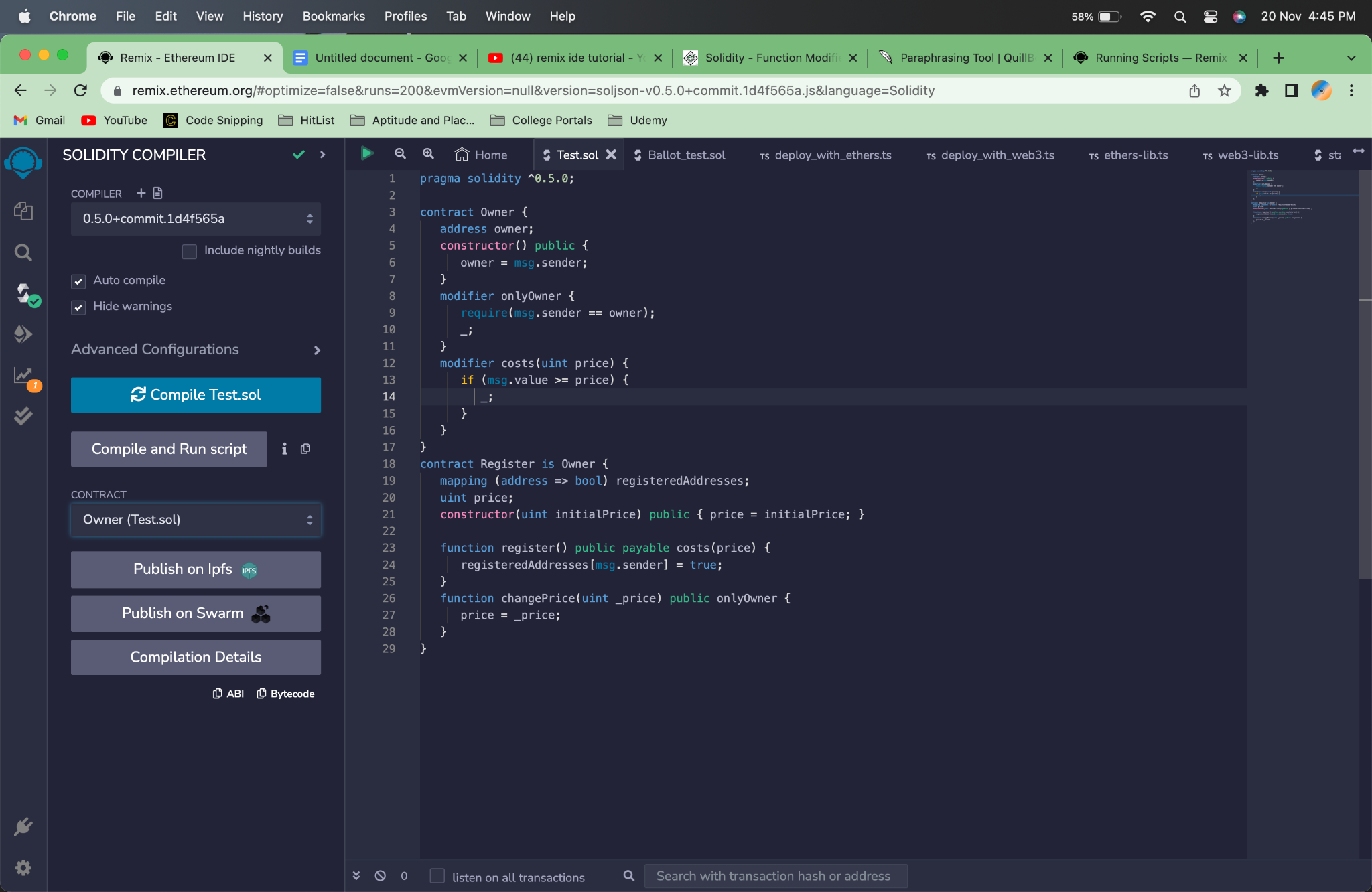
**Implement and demonstrate the use of solidity**

**Functions in Solidity :**



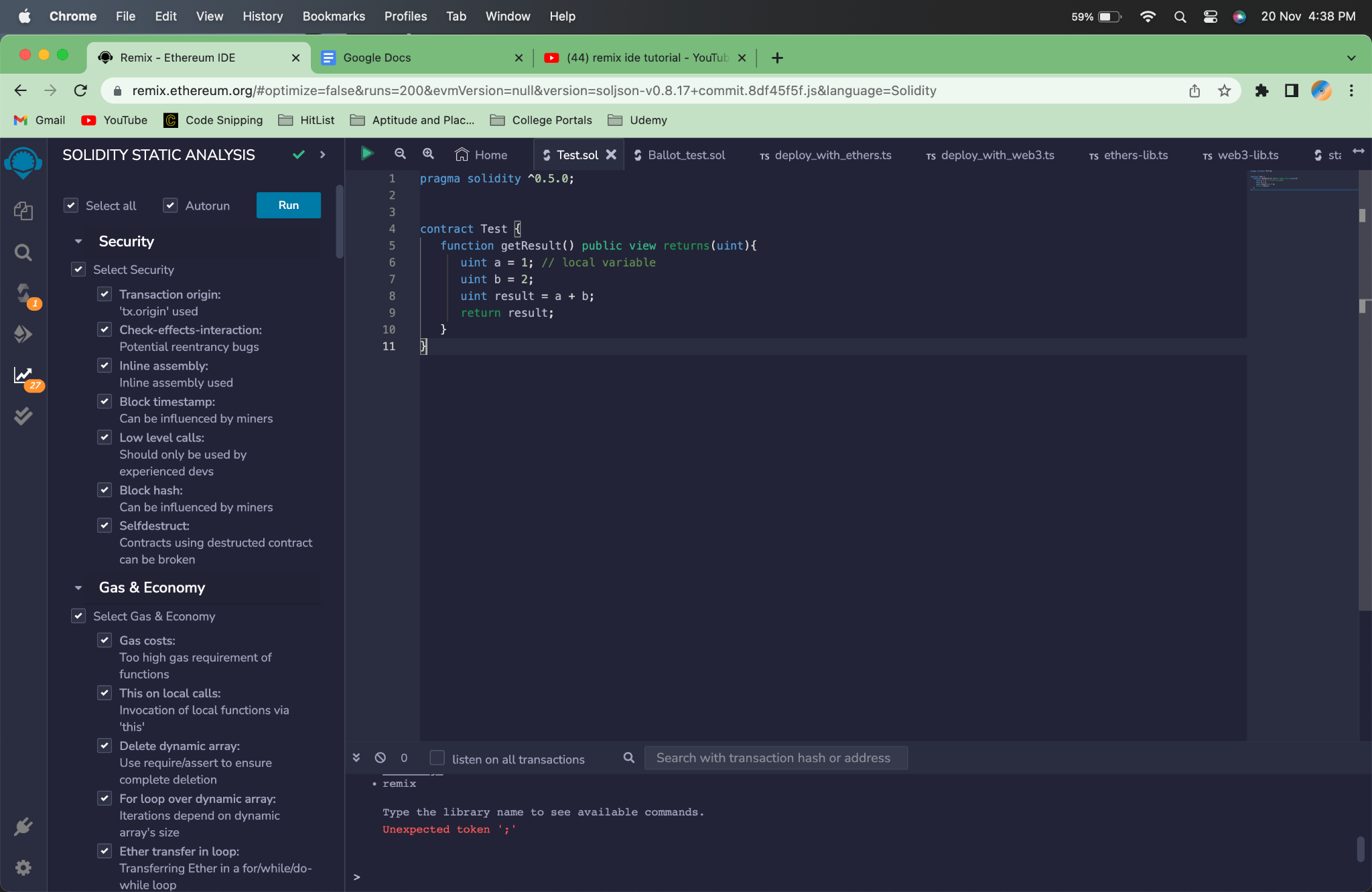
A function must first be defined before it may be used. The most typical approach to define a function in Solidity is to start with the keyword "function", then add a distinctive function name, a list of parameters (which may or may not be empty), and a statement block enclosed in curly brackets.

**Function Modifiers:**



Wherever the special sign "\_;" appears in the definition of a modifier, the function body is appended. Therefore, when invoking this method, if the modifier's condition is satisfied, the function is called; otherwise, an exception is thrown.

**View Functions :**



We can use the same example as above, Functions for the view make sure that they won't change the state. You can declare a function as a view. If any of the following statements are present in the function, the compiler will issue a warning since they are seen as changing the state.

**Pure Functions :**

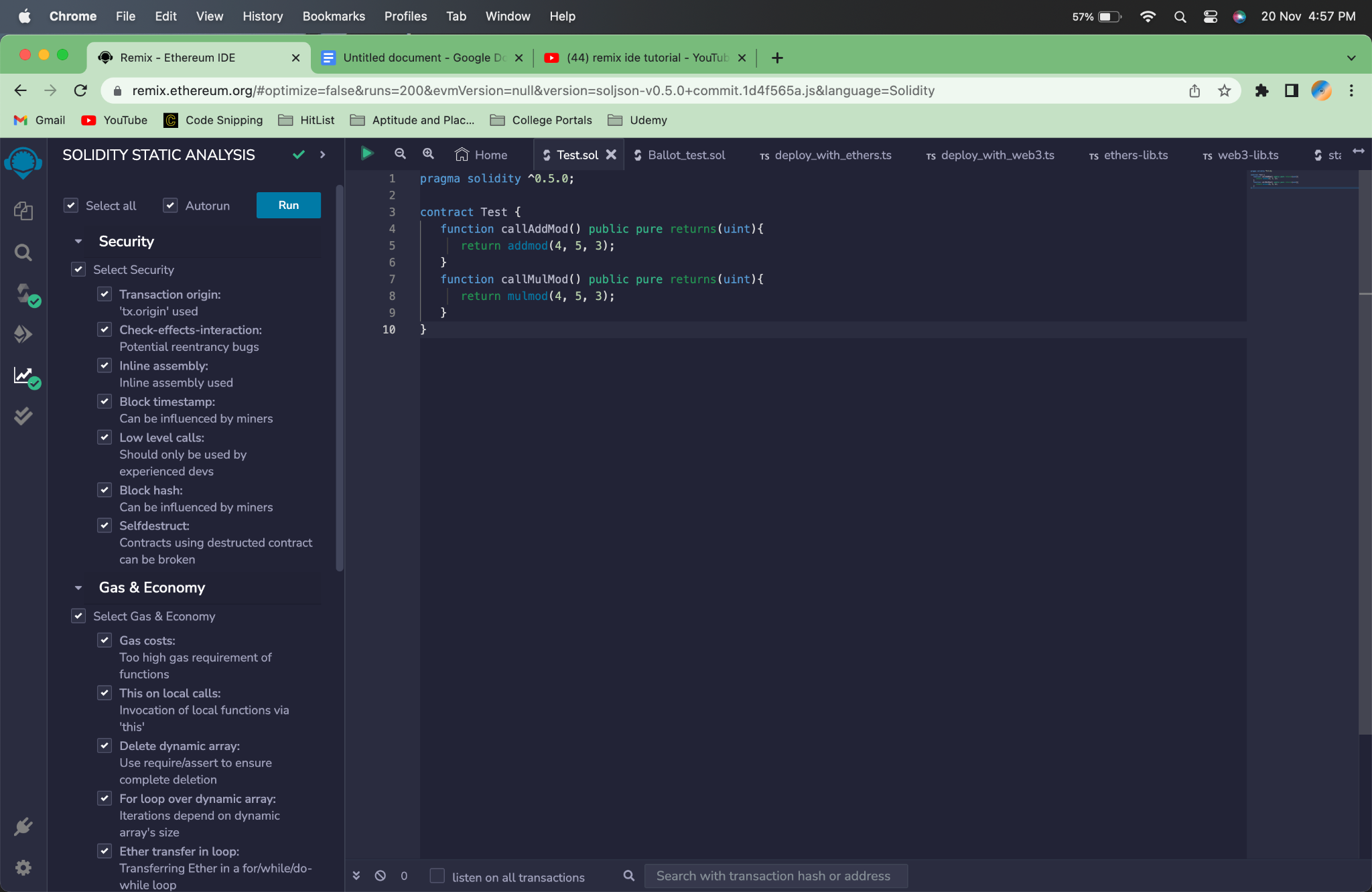
To ensure that they don't read or change the state, pure functions are used. One can declare a function to be pure. The compiler will issue a warning if any of the following statements are present in the function because they are regarded as reading the state.

**Mathematical Functions :**

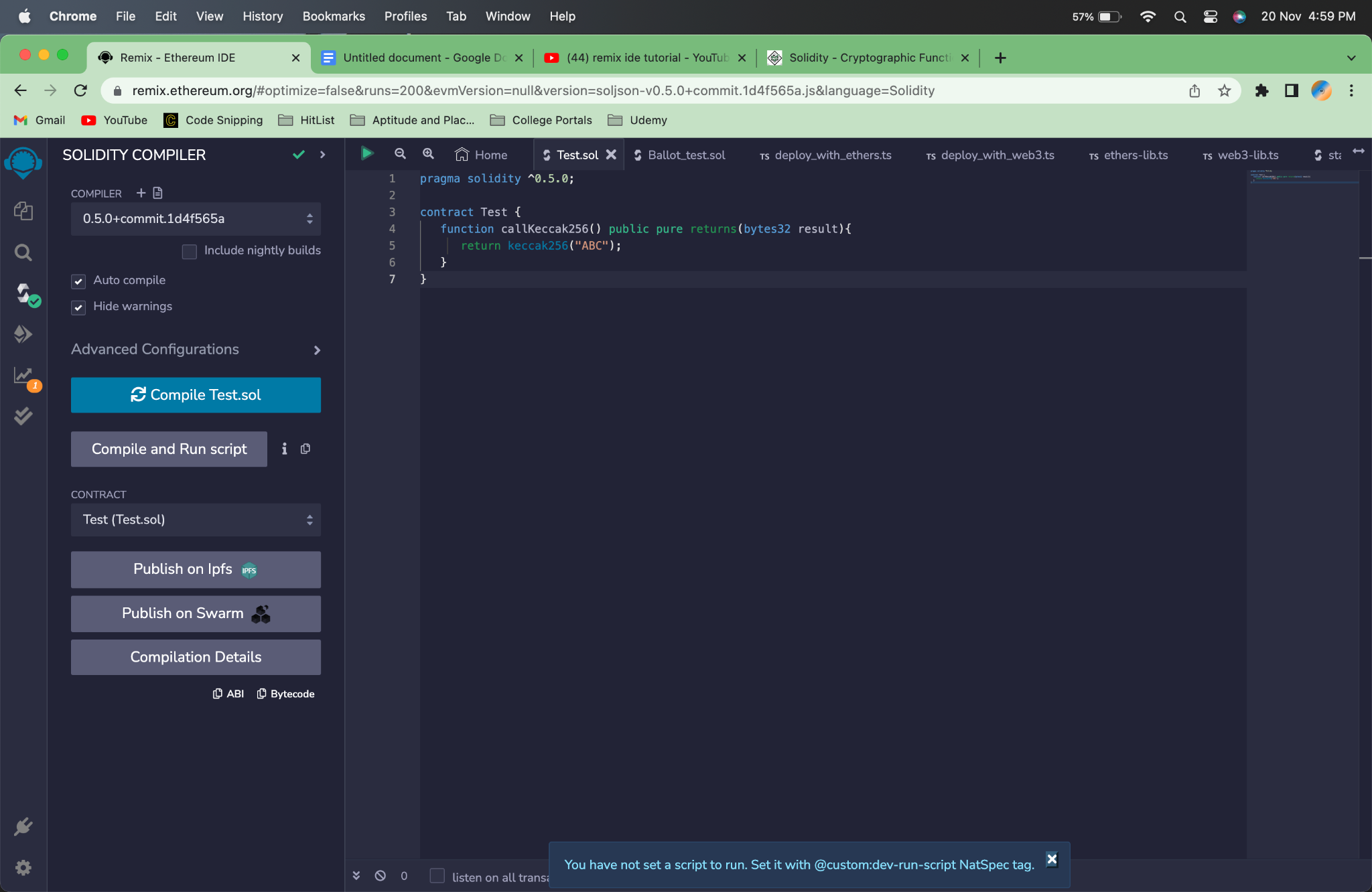
Additionally, solidity has built-in mathematical capabilities. The following techniques are widely used:

The function addmod(uint x, uint y, uint k) computes (x + y)% k with arbitrary accuracy and does not wrap around at 2256.

The function mulmod(uint x, uint y, uint k) computes (x \* y)% k with arbitrary accuracy and does not wrap around at 2256.



# **Cryptographic Functions :**



Solidity provides inbuilt cryptographic functions as well. Following are important methods −

* keccak256(bytes memory) returns (bytes32) − computes the Keccak-256 hash of the input.
* ripemd160(bytes memory) returns (bytes20) − compute RIPEMD-160 hash of the input.
* sha256(bytes memory) returns (bytes32) − computes the SHA-256 hash of the input.
* ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address) − recover the address associated with the public key from elliptic curve signature or return zero on error. The function parameters correspond to ECDSA values of the signature: r - first 32 bytes of signature; s: second 32 bytes of signature; v: final 1 byte of signature. This method returns an address.