Advanced Smart Contracts

Various types in Solidity -

Basic Types		
Name	Notes	Examples
string	Sequence of characters	"Hi there!" "Chocolate"
bool	Boolean value	true false
int	Integer, positive or negative. Has no decimal	0 -30000 59158
uint	'Unsigned' integer, positive number. Has no decimal	0 30000 999910
fixed/ufixed	'Fixed' point number. Number with a decimal after it	20.001 -42.4242 3.14
address	Has methods tied to it for sending money	0x18bae199c8dbae199c8d

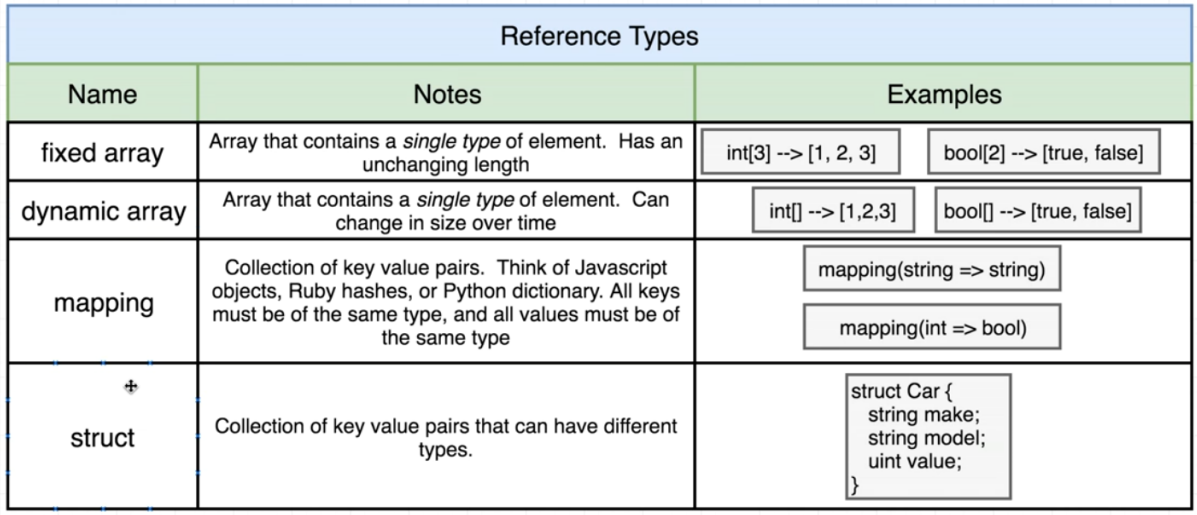
- There are many types of ints as well -
  - The number in front of int represents the number of bits used.
  - We have similar types of uints.

Integer Ranges		
Name	Lower Bound	Upper Bound
int8	-128	127
int16	-32,768	32,767
int32	-2,147,483,648	2,147,483,647
...	...	...
int256	Really, really negative	Really, really big

int == int256

- Reference types in Solidity -

- fixed array → atmost _ number of elements honge array mai.
- For arrays, remember the method which got automatically generated for public instance/storage vars, kinda similar thing happens for arrays as well, difference is that this method accepts 1 argument which is the index of the array and then it returns the element from the array at that index.



Reference Types		
Name	Notes	Examples
fixed array	Array that contains a *single type* of element. Has an unchanging length	int[3] --> [1, 2, 3] bool[2] --> [true, false]
dynamic array	Array that contains a *single type* of element. Can change in size over time	int[] --> [1,2,3] bool[] --> [true, false]
mapping	Collection of key value pairs. Think of Javascript objects, Ruby hashes, or Python dictionary. All keys must be of the same type, and all values must be of the same type	mapping(string => string) mapping(int => bool)
struct	Collection of key value pairs that can have different types.	struct Car { string make; string model; uint value; }

Declaring array in Solidity -

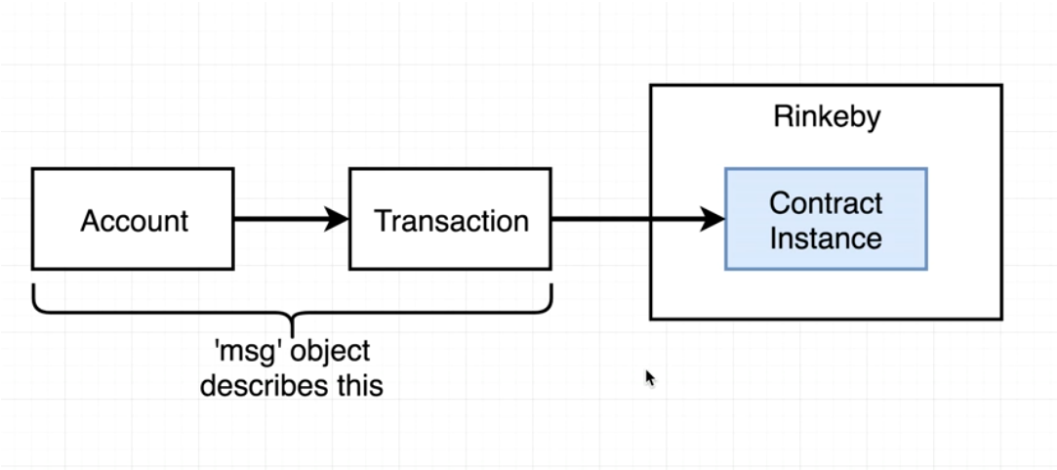
```
address[] public players = new address[](5);
```

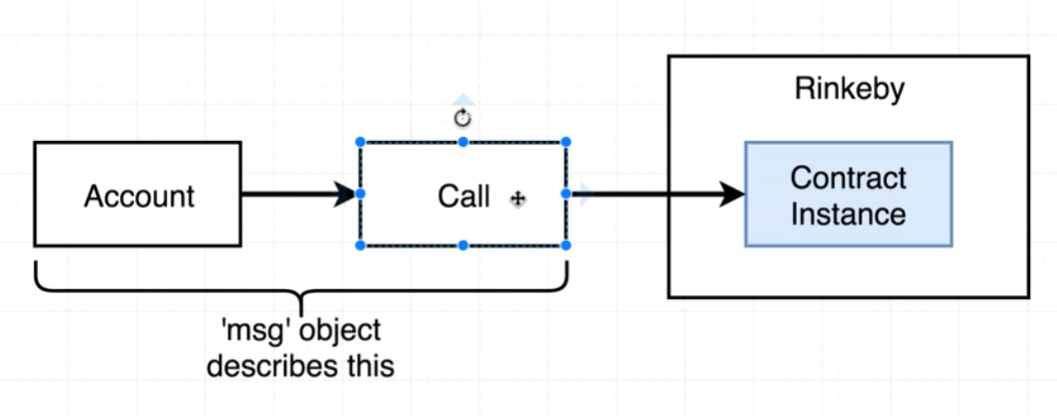
- Above line declares a dynamic array (empty [] on RHS) with **initial size = 5**.
- More precisely, it will create a dynamic array init to = `[0x0000, 0x0000, 0x0000, 0x0000, 0x0000]`.
- `0x0000` is the default address (0 address).

msg Object -

- Whenever we invoke a function, we get access to a msg object which contain info about who invoked the function, and info about the invocation (eg. transaction) as well.

- **This object is globally available inside the function body.**



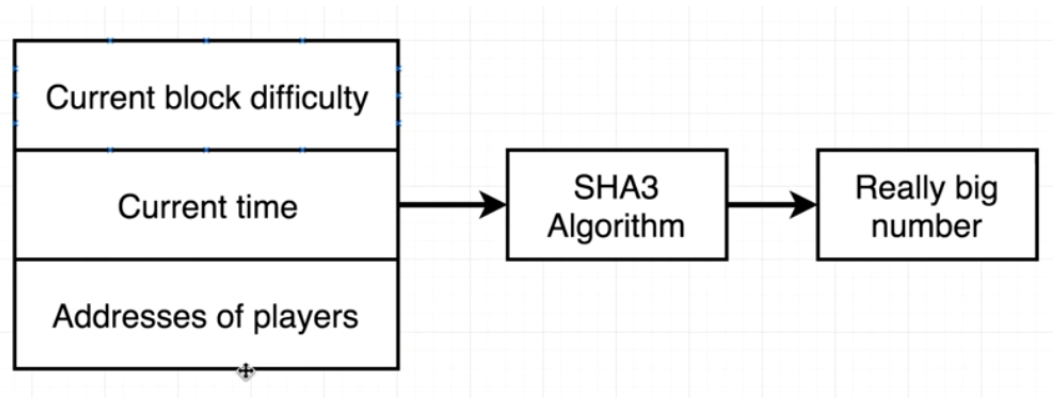


- The msg object looks like the following -

The 'msg' Global Variable	
Property Name	Property Name
msg.data	'Data' field from the call or transaction that invoked the current function
msg.gas	Amount of gas the current function invocation has available
msg.sender	Address of the account that started the current function invocation
msg.value	Amount of ether (in wei) that was sent along with the function invocation

Generating random number in Solidity -

- We will be generating a psuedo random number.
  - Block Difficulty - The time taken to re-hash/solve a block. Integer. Can access it using `block.difficulty`.
  - Current Time - Can access it using `now`.



Transfering ether to an address -

- The address of an account in solidity is an object on which we can call certain methods one of which is `transfer()`, using this we can transfer some ether to that address.

Function Modifiers -

- Implements DRY.
- Used to run some code before the body of a function runs
- Syntax of function modifiers -

```
modifier <name-of-func-modifier>(){
  code ……
  _;
}
```

- Using function modifiers -

```
function <name-of-function>(...) <access-modifier> <name-of-func-modifier> ... {
  code ...
}
```

Misc -

- When we want someone to send some ether upon calling a method from our smart contract then we mark this method as **payable.** Also, we can enforce that a particular amount of money is to be sent upon a function call by using the require() global function in Solidity.
- We can use the sha3 algo in sol code just by using the `sha3` global function.
- To reference all the ether stored in an instance of a contract we use `this.balance`. `this` refers to the current instance of the contract.
- We enforce some level of security using `require()` statements.
- Converting value from one unit to another using web3 lib.
  - Converting Ether to Wei -

```
web3.utils.toWei("<value>", "ether")
```

- assert.equal(value it should be, value it is);