G. N. KHALSA COLLEGE

AUTONOMOUS

(UNIVERSITY OF MUMBAI)

MUMBAI-400019

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

Exam Seat No. 242

CERTIFIED that the practicals, assignment duly signed, were performed by

Mr. ___**Sahil Kilje**___Roll No. **242** of M.Sc Part II class in the Computer Science Laboratory of G. N. Khalsa College, Mumbai during academic year 2022- 2023.

He has completed the course of Laboratory assignments in Computer Science

as contained in the course prescribed by the University of Mumbai.

Sign. Of the Student	**Head of Dept.**
Date_________________	**Computer Science**
	Date_________________
Professor-in-charge	**Sign of Examiner's**
1)________________	1)________________
Date_____________	Date______________
2)______________	2)________________
Date____________	Date_______________

Sahil Kilje
Roll No. 242

WEB TECHNOLOGIES

INDEX

6	Implement and demonstrate the use of following in solidity: 1. Functions 2. View Functions 3. Pure Functions 4. Payable Function.	19-11-22	41	
7	Implement and demonstrate the use of following in solidity: 1. Function Overloading 2. Mathematical Functions 3. Cryptographic Functions	19-11-22	45	
8	Implement and demonstrate the use of following in solidity: 1. Pubic 2. Private 3. Internal 4. External 5. Contract 6. Constructors	19-11-22	48	

Sahil Kilje
Roll No. 242

PRACTICAL NO: 01

AIM: Installing Jupyter Notebook using Anaconda.

Theory:

- **Anaconda:**

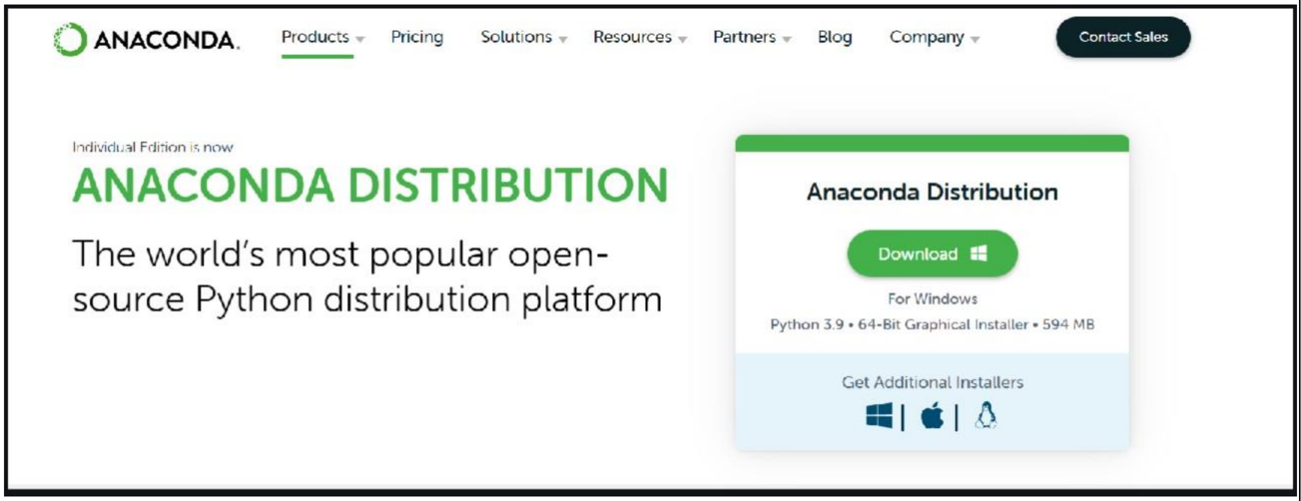
  Anaconda is an open-source software that contains Jupyter, spyder, etc that are used for large data processing, data analytics, heavy scientific computing. Anaconda works for R and python programming language. Spyder(sub-application of Anaconda) is used for python.
  Opencv for python will work in spyder. Package versions are managed by the package management system called conda.

- **Jupyter Notebook:**

  Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. Jupyter has support for over 40 di‰erent programming languages and Python is one of them.

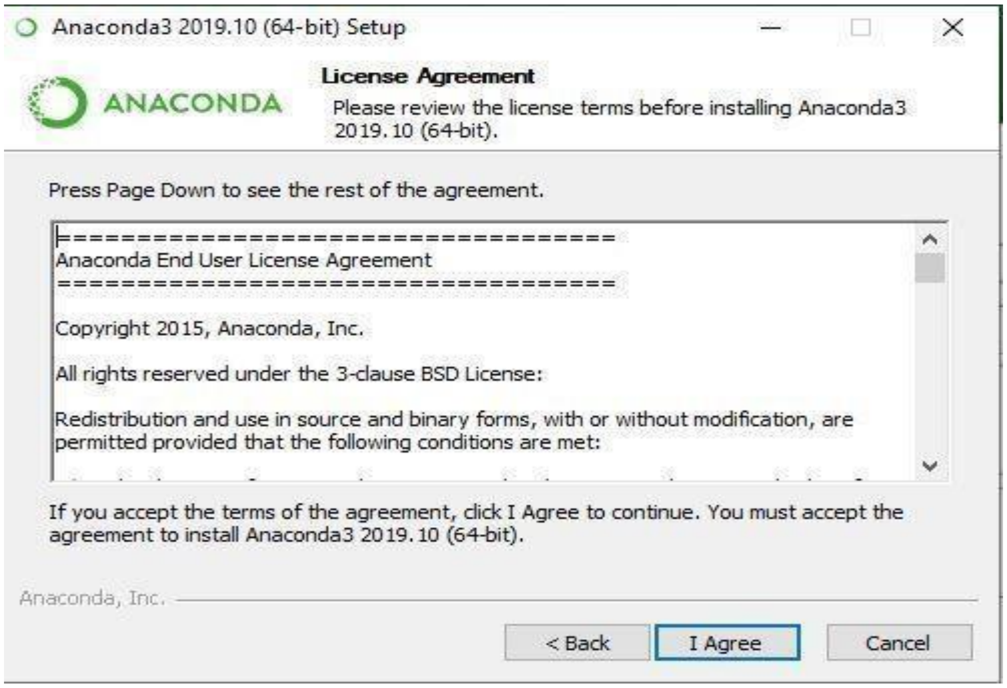
a) Steps to install Anaconda:

1. Head over to anaconda.com and install the latest version of Anaconda. Make sure to download the "Python 3.7 Version" for the appropriate architecture. There are versions available for Linux and mac distributions.



Individual Edition is now
ANACONDA DISTRIBUTION
The world's most popular open-source Python distribution platform

For Windows
Python 3.9 • 64-Bit Graphical Installer • 594 MB


Anaconda3 2019.10 (64-bit) Setup

License Agreement

Please review the license terms before installing Anaconda3 2019.10 (64-bit).

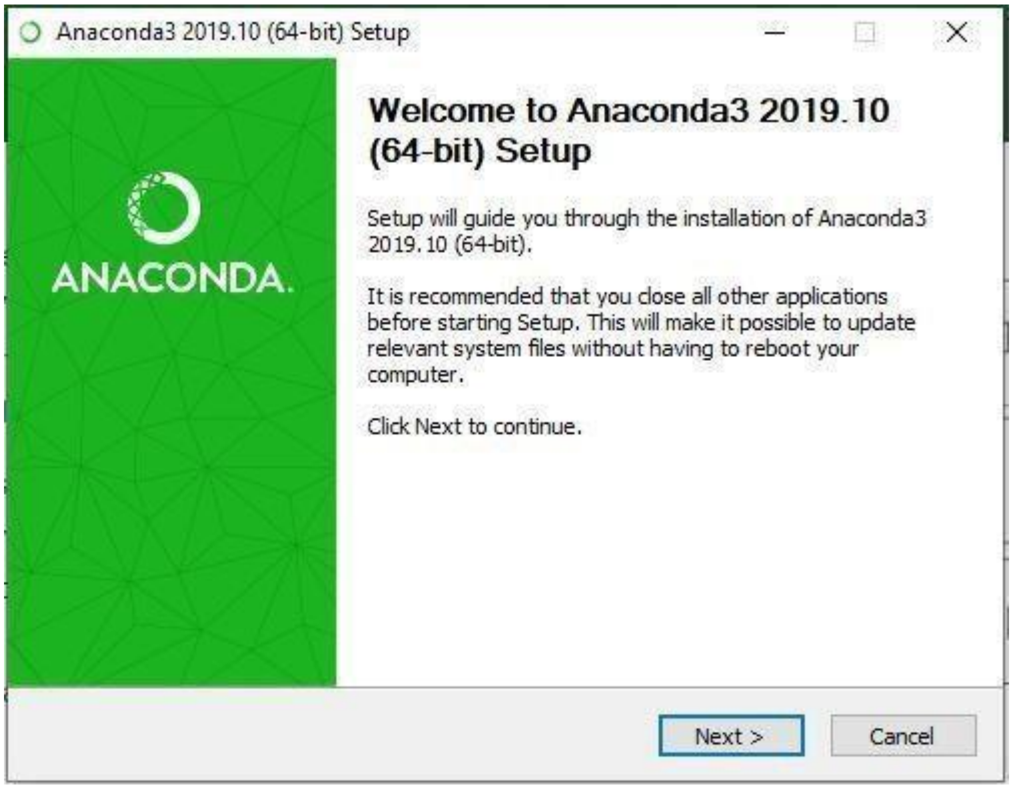
Press Page Down to see the rest of the agreement.

```
===================================
Anaconda End User License Agreement
===================================

Copyright 2015, Anaconda, Inc.

All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without modification, are
permitted provided that the following conditions are met:
```

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2019.10 (64-bit).

Anaconda, Inc.

2. Now start with the installation process, double click to open .exe file and click next and follow the instructions shown in images.


Anaconda3 2019.10 (64-bit) Setup

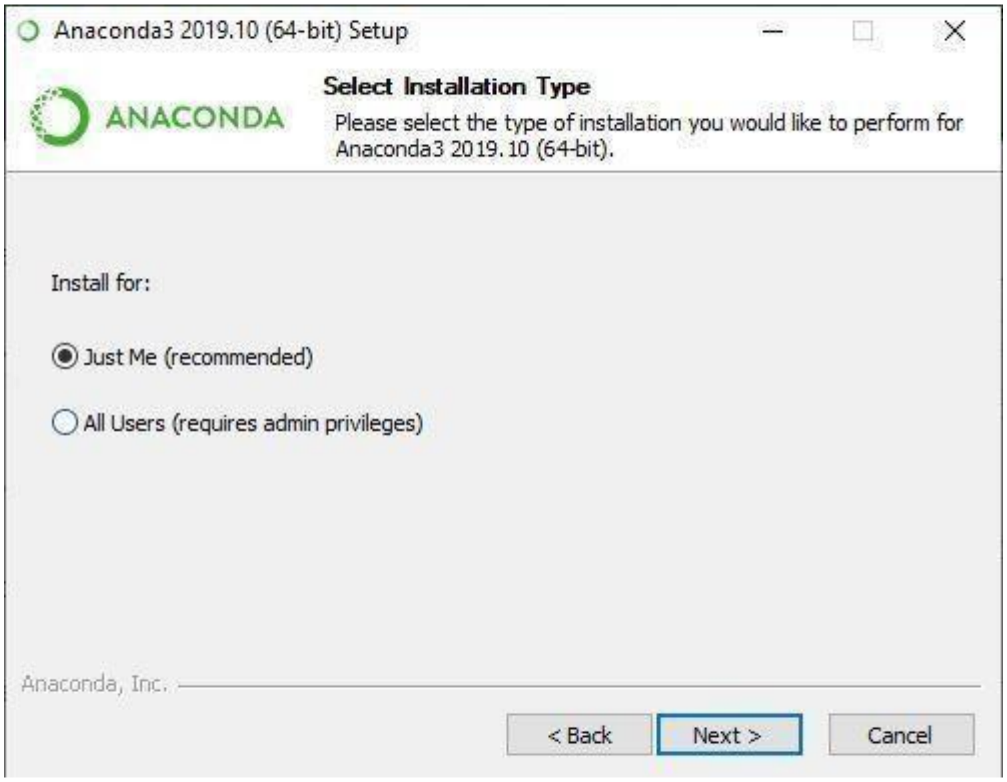
Welcome to Anaconda3 2019.10 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 2019.10 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

3. click on “I agree”.
4. Select Just me.


Anaconda3 2019.10 (64-bit) Setup

Select Installation Type

Please select the type of installation you would like to perform for Anaconda3 2019.10 (64-bit).

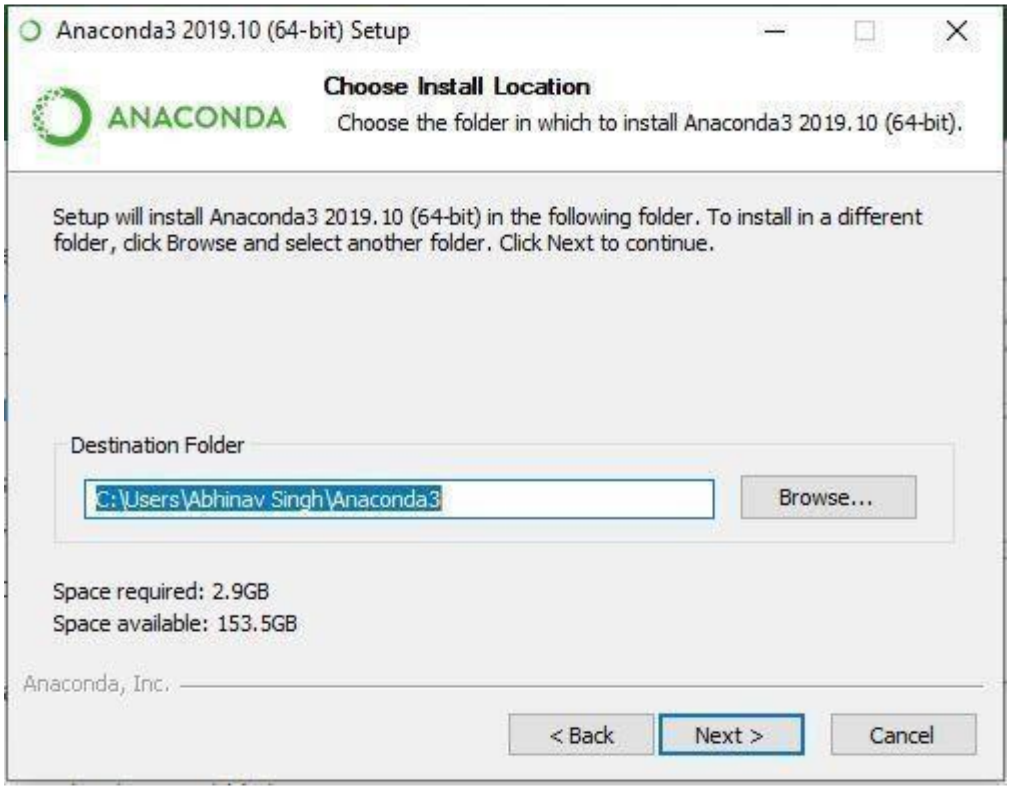
Install for:

[✓ Just Me (recommended)]

All Users (requires admin privileges)

Anaconda, Inc.

5. Choose Installation Directory. It is advisable to use the system OS directory only i.e., C.


Anaconda3 2019.10 (64-bit) Setup

Choose Install Location

Choose the folder in which to install Anaconda3 2019.10 (64-bit).

Setup will install Anaconda3 2019.10 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

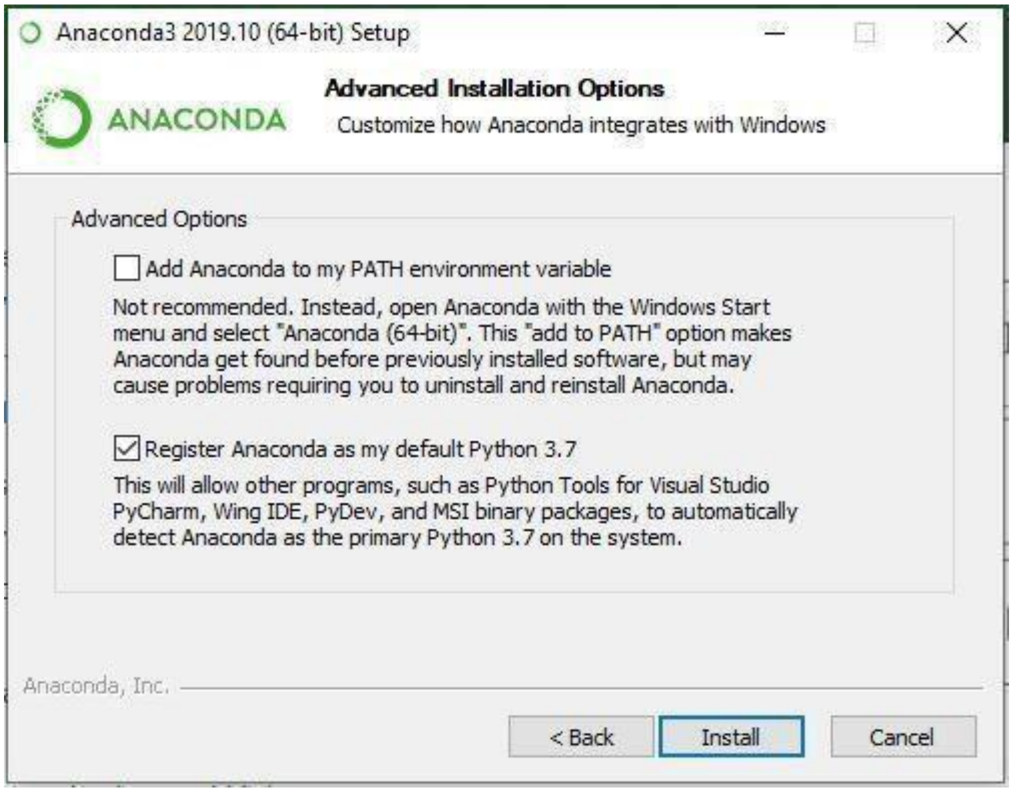
Destination Folder

C:\Users\Abhinav Singh\Anaconda3

Space required: 2.9GB
Space available: 153.5GB

Anaconda, Inc.

6. Advance Installation Option:



Anaconda3 2019.10 (64-bit) Setup

Advanced Installation Options

Customize how Anaconda integrates with Windows

Advanced Options

☐ Add Anaconda to my PATH environment variable

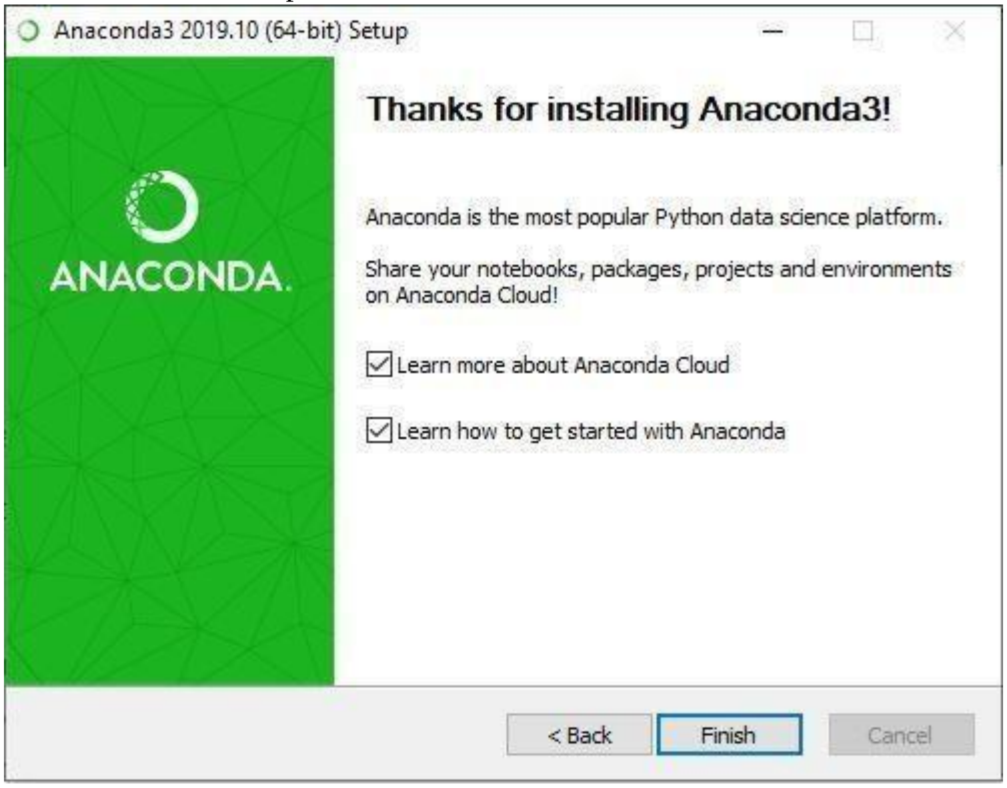
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☑ Register Anaconda as my default Python 3.7

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.

Anaconda, Inc.

7. Finish the installation process.



Anaconda3 2019.10 (64-bit) Setup

Thanks for installing Anaconda3!

Anaconda is the most popular Python data science platform.

Share your notebooks, packages, projects and environments on Anaconda Cloud!

☑ Learn more about Anaconda Cloud

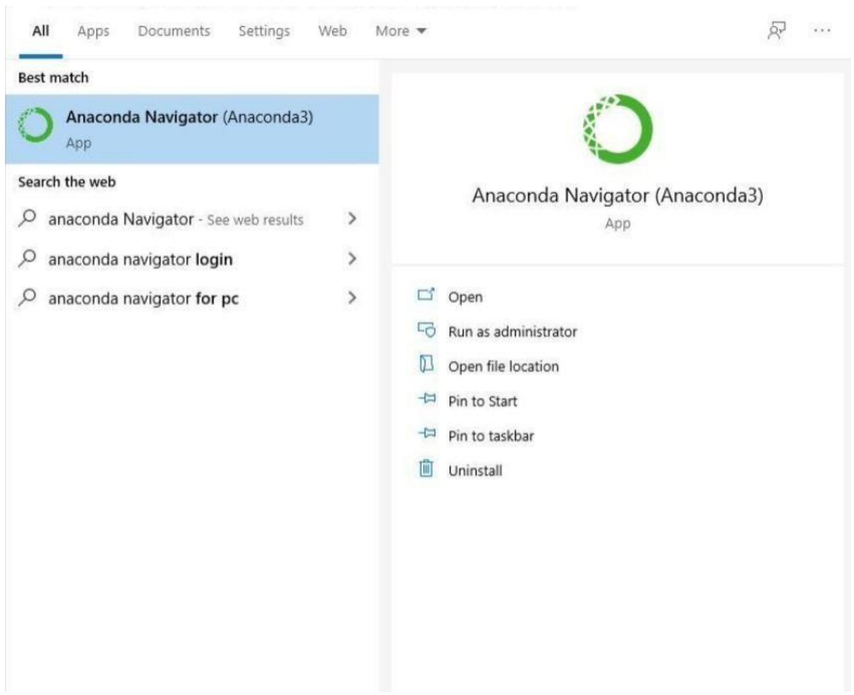
☑ Learn how to get started with Anaconda

b) Installing Jupyter Notebook using Anaconda:

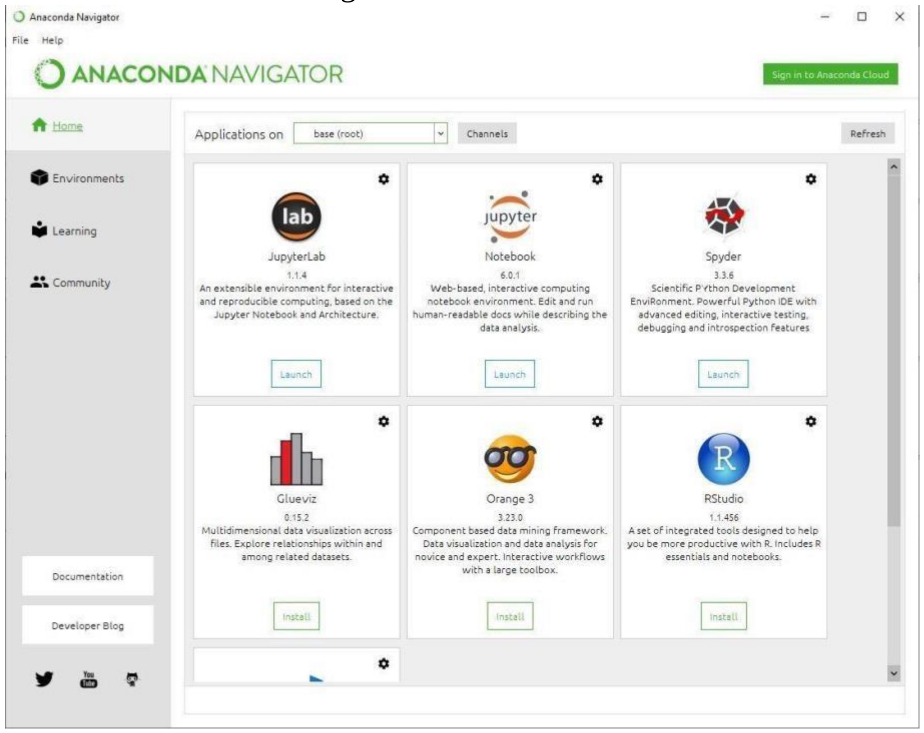
Once the installation process is done, Anaconda can be used to perform multiple operations.

Sahil Kilje
Roll No. 242

1. To begin using Anaconda, search for Anaconda Navigator from the Start Menu in Windows



2. Launch Anaconda Navigator:



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

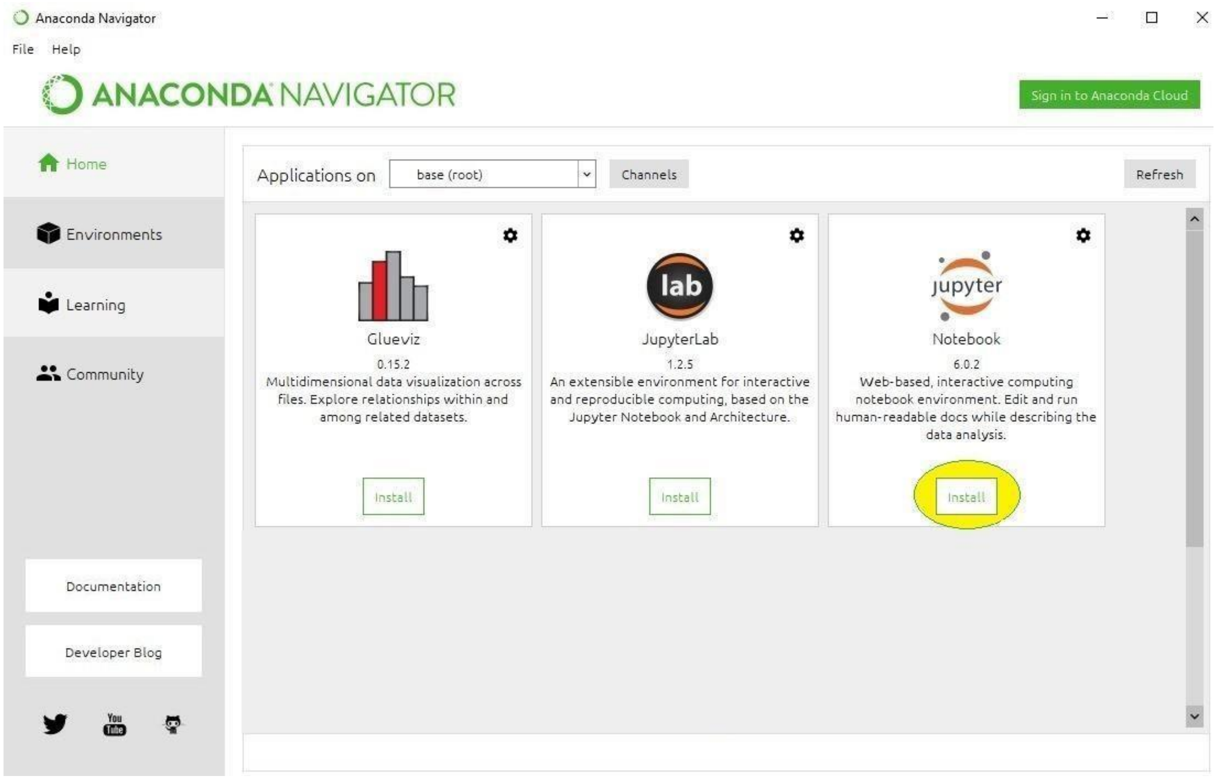
Multidimensional data visualization across files. Explore relationships within and among related datasets.

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

3. Click on Install Jupyter Notebook Button:

Sahil Kilje
Roll No. 242

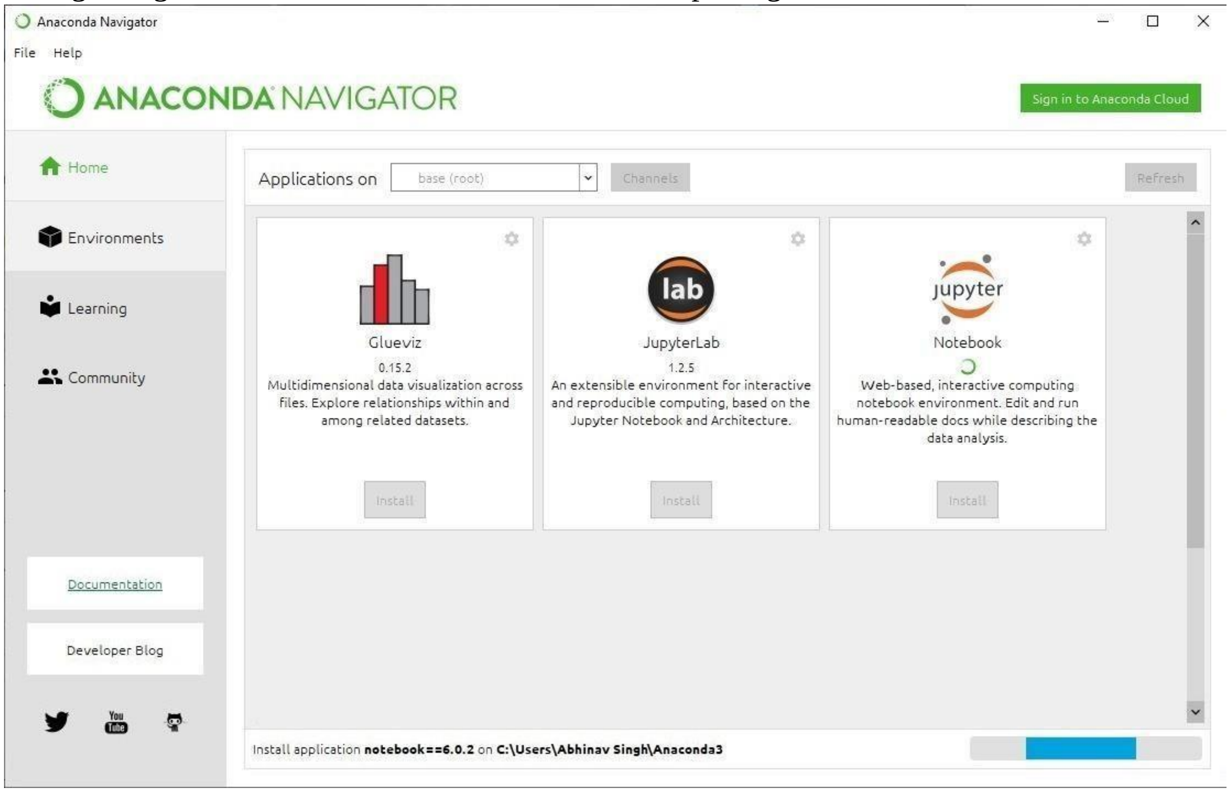


Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.

JupyterLab
1.2.5
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.0.2
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

4. Beginning the Installation, it takes time to load the packages.



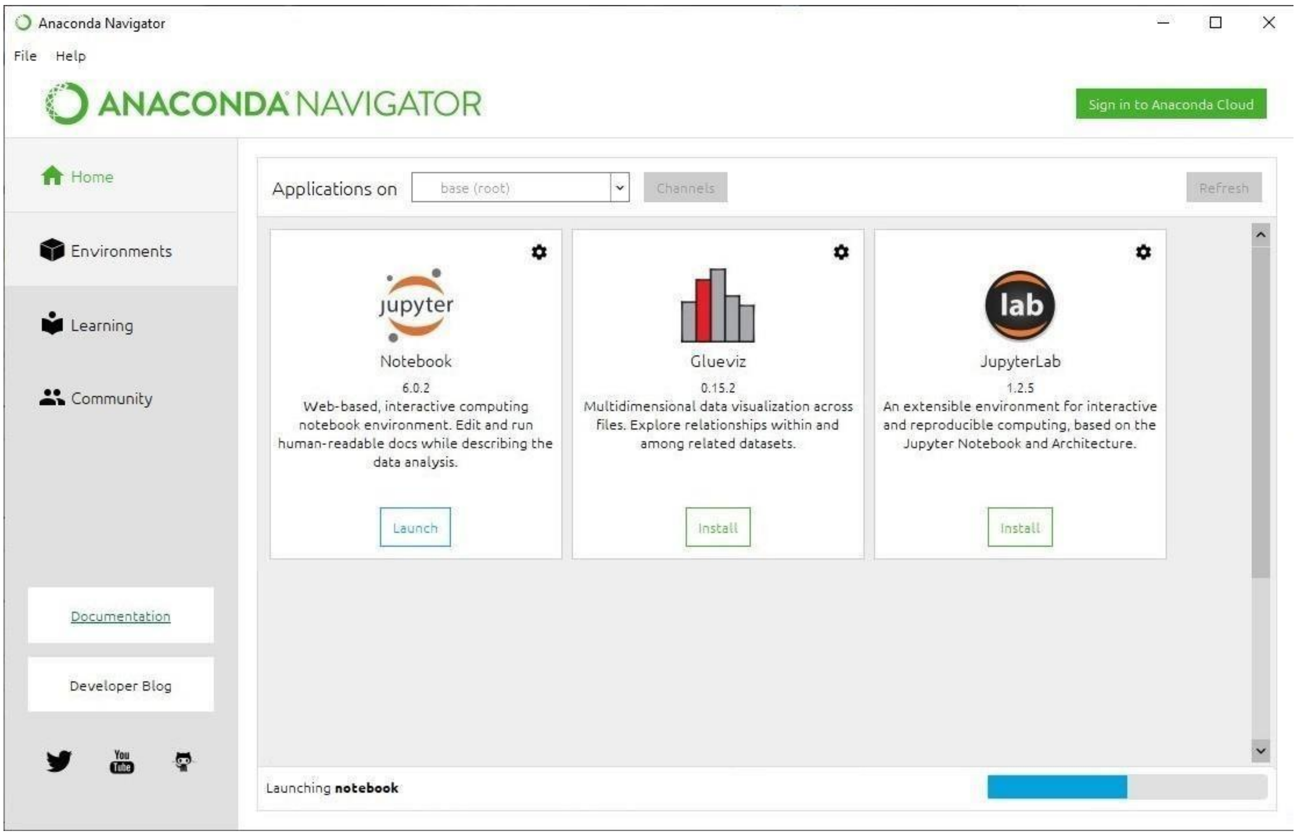
Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.

JupyterLab
1.2.5
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Install application **notebook==6.0.2** on **C:\Users\Abhinav Singh\Anaconda3**

5. After the installation is completed, launch the Jupyter notebook, and then you are good to use Jupyter Notebook.

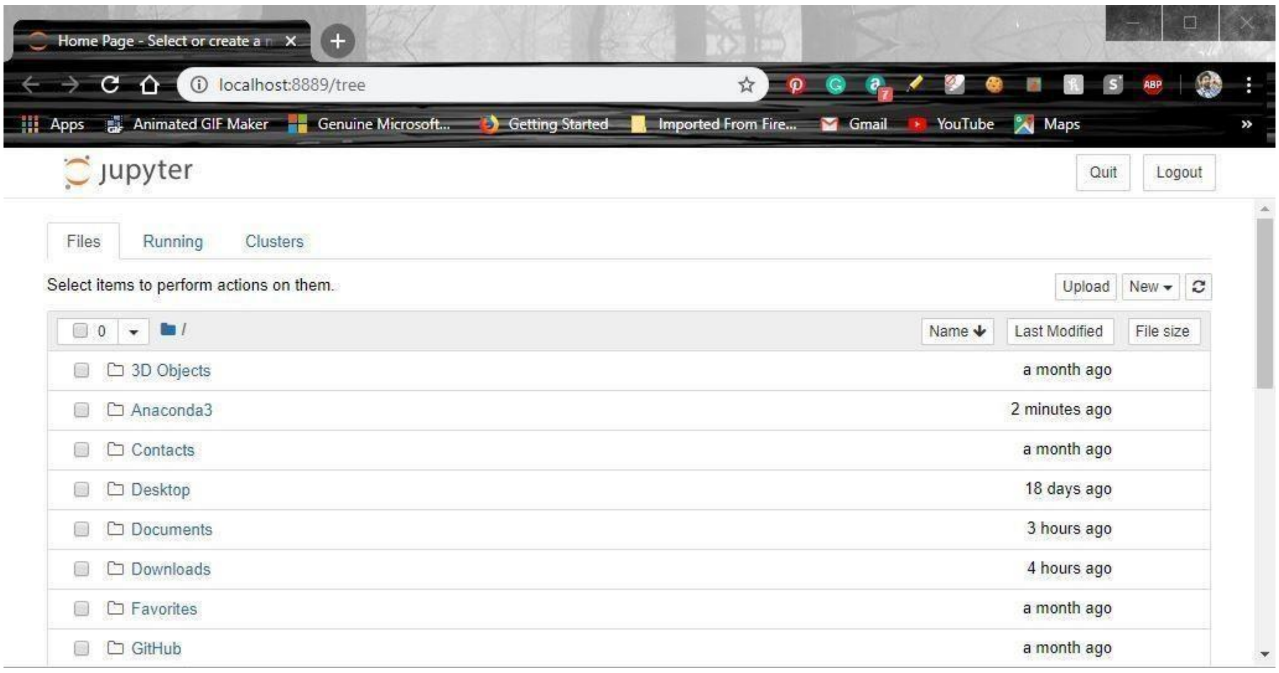


Notebook
6.0.2
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.

JupyterLab
1.2.5
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launching **notebook**



Select items to perform actions on them.

Name	Last Modified
3D Objects	a month ago
Anaconda3	2 minutes ago
Contacts	a month ago
Desktop	18 days ago
Documents	3 hours ago
Downloads	4 hours ago
Favorites	a month ago
GitHub	a month ago

Sahil Kilje
Roll No. 242

PRACTICAL NO: 02

AIM:

2.1 transaction class to send and recieve money

2.2 create public and private key

THEORY:

What is public key?

an encryption technique uses a pair of keys (public and private key) for secure data communication. In the pair of keys, the public key is for encrypting the plain text to convert it into ciphertext.

The public key can be shared without compromising the security of the private one. All asymmetric key pairs are unique, so a message encrypted with a public key can only be read by the person who has the corresponding private key. The keys in the pair have much longer than those used in symmetric cryptography. So, it is hard to decipher the private key from its public counterpart. Many of us, heard about RSA, which is the most common algorithm for asymmetric encryption in use today.

What is private key?

the private key is used for decrypting the ciphertext to read the message.

The private key is given to the receiver while the public key is provided to the public. Public Key Cryptography is also known as asymmetric cryptography.

What is cryptography?

Cryptography is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents.

The term is derived from the Greek word kryptos, which means hidden.

It is closely associated to encryption, which is the act of scrambling ordinary text into what's known as ciphertext and then back again upon arrival.

What is blockchain?

Blockchain can be defined as a chain of blocks that contains information.

The technique is intended to timestamp digital documents so that it’s not possible to backdate them or temper them.

Sahil Kilje
Roll No. 242

The purpose of blockchain is to solve the double records problem without the need for a central server.
The blockchain is used for the secure transfer of items like money, property, contracts, etc, without requiring a third-party intermediary like a bank or government. Once data is

```
pip install pycryptodome
```

recorded inside a blockchain, it is very difficult to change it.

In [1]:

```
Collecting pycryptodome
  Downloading pycryptodome-3.15.0-cp35-abi3-win_amd64.whl (1.9 MB)


 Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.15.0
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```
#2.1 transaction class to send and recieve money
#2.2 create public and private key
```

In [1]:

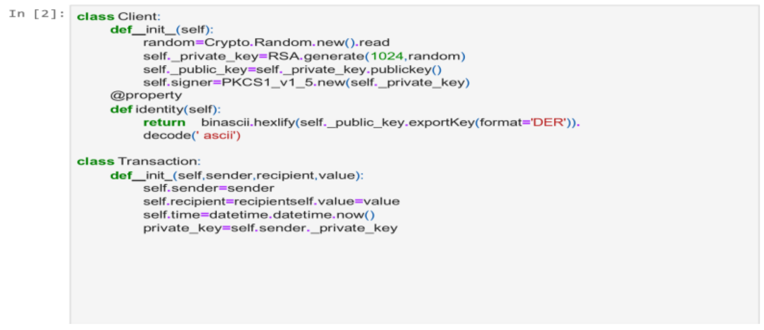
```
pip install cryptography
```

```
 Requirement already satisfied: cryptography in c:\users\jagra\anaconda3\lib\site-p  ackages
(3.4.8)
Requirement already satisfied: cffi>=1.12 in c:\users\jagra\anaconda3\lib\site-pac
kages (from cryptography) (1.15.0)
Requirement already satisfied: pycparser in c:\users\jagra\anaconda3\lib\site-pack
ages (from cffi>=1.12->cryptography) (2.21)
```

```
import binascii import numpy as np import
pandas as pd import Crypto import
Crypto.Random from Crypto.Hash import
SHA from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
import collections import datetime
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [1]:



In [2]:

```
class Client:
      def__init_(self):
            random=Crypto.Random.new().read
            self._private_key=RSA.generate(1024,random)
            self._public_key=self._private_key.publickey()
            self.signer=PKCS1_v1_5.new(self._private_key)
      @property
      def identity(self):
            return   binascii.hexlify(self._public_key.exportKey(format='DER')).
            decode(' ascii')

class Transaction:
      def__init_(self,sender,recipient,value):
            self.sender=sender
            self.recipient=recipientself.value=value
            self.time=datetime.datetime.now()
            private_key=self.sender._private_key
```

```
    def to_dict(self):
        if self.sender=="Genesis":
            identity="Genesis"
        else:
            identity=self.sender.identity
        return collections.OrderedDict({ 'sender':identity,
            'recipient':self.recipient, 'value':self.value,
            'time':self.time })

    def sign_transaction(self):
            signer=PKCS1_v1_5.new(private_key)
            h=SHA.new(str(self.to_dict()).encode('utf8'))
            return  binascii.hexlify(signer.sign(h)).decode("ascii")
```

In [3]:
```
Dinesh=Client()
Ramesh=Client()
```

In [4]:
```
t=Transaction(Dinesh,Ramesh.identity,5.0)
```

In [5]:
```
print(Dinesh.identity)
```

```
30819f300d06092a864886f70d010101050003818d0030818902818100cd0e692ccb3a62c792cd4741
bce2fa19b5e1321d7e34985c4a99c9554a318ff9c759cd8fab2972319041b00d92a1f9f0a428cec3cd
e6ae47af808b23b57206204d16c98bfbdace370bc7753c234e211e0252359ac3e6a62f294105e7c6ec
a6ea3e061870a36259cb2cab992c5d6d446a7c07d74024c404892b4fa8ad388e3b1f0203010001
```

In [6]:
```
signature=t.sign_transaction() print(signature)
```

```
250f71bd3e9d2e2ab1ed183057508b509a80ebc3b7f46b852f15c4ebcedab7dcadf3595cc6af884ecb
8ed6ac24b7e4c40ed8989bb50699cee825a707bbd95dd611dc5a12c1dd140e2c4467d665dd0cedab2f
514320e784c5c7a9aafd93f05c2cd1780cebe7134aeb9c7ea19b74482c0f0429b8bbf91fd43056e13d
73ec9518ba
```

In [ ]:

Sahil Kilje
Roll No. 242

PRACTICAL NO: 03

AIM: Create multiple transactions and display them

THEORY:

What is Miner?

1. A computer of group of computers that do bitcoin transactions (adding new transactions or verifying blocks created by other miners. Miners are rewarded with transactions fees. Learn more in: Has Bitcoin Achieved the Characteristics of Money?
2. Mining is a process of adding transactions to the large distributed public ledger of existing transactions which are known as the block chain. The person involved in mining is hence called a miner. Learn more in: Blockchain for Islamic Financial Services Institutions: The Case of Sukuk Financing
3. The individual who executes mining of cryptocurrency. Learn more in:

Evolution of Cryptocurrency: Analysing the Utility, Legality, and Regulatory Framework in India

4. Miners validate new blockchain transactions and record them on the blockchain. Miners compete to solve a difficult mathematical problem based on a cryptographic hash algorithm. Learn more in: End-to-End Tracing and Congestion in a Blockchain: A Supply Chain Use Case in Hyperledger Fabric
5. Miner is an actor who participates in cryptocurrency transactions, and in turn, plays a crucial role both in creating new cryptocurrencies and in verifying transactions on the blockchain. It adds new blocks to the existing chain, and ensures that these additions are accurate. Learn more in: Blockchain Technology: Concepts, Components, and Cases

What is Role of Miner?

Within the bitcoin networks, there are a group of people known as Miners. In miners, there was a process and confirm transactions. Anybody can apply for a minor, and you could run the client yourself. However, these minors use very powerful computers that are specifically designed to mine bitcoin transaction. They do this by actually solving math problems and resolving cryptographic issues because every transaction needs to be cryptographically encoded and secured. These mathematical problems ensure that nobody is tampering with that data. Additionally, for this task, the minors are paid in bitcoins, which is the key component in bitcoin. In Bitcoin, you cannot create money as like you create regular fiat currencies such as Dollar, Euro, and Yuan. The bitcoin is created by rewarding these minors for their work in solving the mathematical and cryptographical problems.

Sahil Kilje
Roll No. 242

In [1]:
```
pip install pycryptodome
```

```
Requirement already satisfied: pycryptodome in c:\users\jagra\anaconda3\lib\site-p
ackages (3.15.0)
```

```
#AIM: create multiple transactions and display them
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
In [2]:
```
pip install cryptography
```

```
Requirement already satisfied: cryptography in c:\users\jagra\anaconda3\lib\site-p
ackages (3.4.8)
```

In [3]:
```
import binascii import numpy as np import
pandas as pd import Crypto import
Crypto.Random from Crypto.Hash import
SHA from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
import collections import datetime
```

In [4]:
```
class Client:
        def__init_(self):
                random=Crypto.Random.new().read
                self._private_key=RSA.generate(1024,random)
                self._public_key=self._private_key.publickey()
                self.signer=PKCS1_v1_5.new(self._private_key)
        @property
        def identity(self):
                return  binascii.hexlify(self._public_key.exportKey(format='DER'))
                .decode(' ascii')

class Transaction:
        def__init_(self,sender,recipient,value):
                self.sender=sender
                self.recipient=recipientself.value=value
                self.time=datetime.datetime.now()
        def to_dict(self):
                if  self.sender=="Genesis":
                        identity="Genesis"
                else:
                        identity=self.sender.identity
                key)
                h=SHA.new(str(self.to_dict()).encode('utf8'))
                return  binascii.hexlify(signer.sign(h)).decode("ascii")
return collections.OrderedDict({ 'sender':identity,
                     'recipient':self.recipient,
                     'value':self.value, 'time':self.time
                 })

            def sign_transaction(self):
                  private_key=self.sender._private_key
                  signer=PKCS1_v1_5.new(private_key)
                  h=SHA.new(str(self.to_dict()).encode('utf8')) return
                  binascii.hexlify(signer.sign(h)).decode("ascii")
```

```
In [5]: Dinesh=Client()
           Ramesh=Client()
```

```
In [6]: t=Transaction(Dinesh,Ramesh.identity,5.0)
```

```
In [7]: signature=t.sign_transaction()
           print(signature)
a001f7799dd45f95d5e222a638cab7c265c42b860f924345d5b85f79c5f28f78ef116e8d1dec47a2e2
          35cc243acaee73f784ae76ba0c74a7a271fd8f0f2dab1d107768a2752e3d99091adb1395f108b77b07
          b39770d25b87a656d522d000be1df9db4651fa08242dc881e7ac5b28940d368b57f6af56f2a819ae6a 5262df930e
```

```
In [32]: def display_transaction(transaction): dict=transaction.to_dict()
             print("sender : "+dict['sender'])
             print(" ---------------------------------------- ")
             print("recipient : "+dict['recipient'])
             print(" ---------------------------------------- ")
             print("value : "+str(dict['value']))
             print(" ---------------------------------------- ")
             print("time : "+str(dict['time']))
             print(" ---------------------------------------- ")
```

```
In [33]: transactions=[]
```

```
In [34]: Dinesh=Client()
         Ramesh=Client()
         Seema=Client()
         Vijay=Client()
```

In [35]:
```
t1= Transaction(Dinesh,Ramesh.identity,18.0)
```

```
t1.sign_transaction() transactions.append(t1)
```

In [36]:

In [37]:
```
display_transaction(t1)
```

```
 sender : 30819f300d06092a864886f70d010101050003818d0030818902818100d9463eb6d85f06c
fd96ea3b0a759bbf21007f252f307b79275004f8e13407b80d033ccd2b3537055070ad819db09fcda8
207234a23a1c2a7282cd6277c92a92beca7363fa30748fc293cd2b0cb733ecc3c57e4370af7f8940d8
ec72bef8ffc44b01da82a35bf3a00f8bc84a42816f0eaac79a25655dc36ac31650cf26e65d91102030
10001

--------------------------------------------------------------------------------------

recipient : 30819f300d06092a864886f70d010101050003818d0030818902818100da943eb53b30
672c92faf5776af7b85d78cd15477b6244869b60e2fb91a5b09486057a74a7870bf9cea316ba856e75
94530d3a341b9b2b4728c40fe3f6ad54eaafe78cabab165125d1efeb59e2958bd9c322e7fe86db4a2b
0fed6298096469abbc5f840bba9816772faab5896f63625f1f3085dada2583eb010c66870d06bf5502
03010001

--------------------------------------------------------------------------------------

value : 18.0

--------------------------------------------------------------------------------------

time : 2022-10-22 09:07:56.851364
--------------------------------------------------------------------------------------
```

In [
]:

Sahil Kilje
Roll No. 242

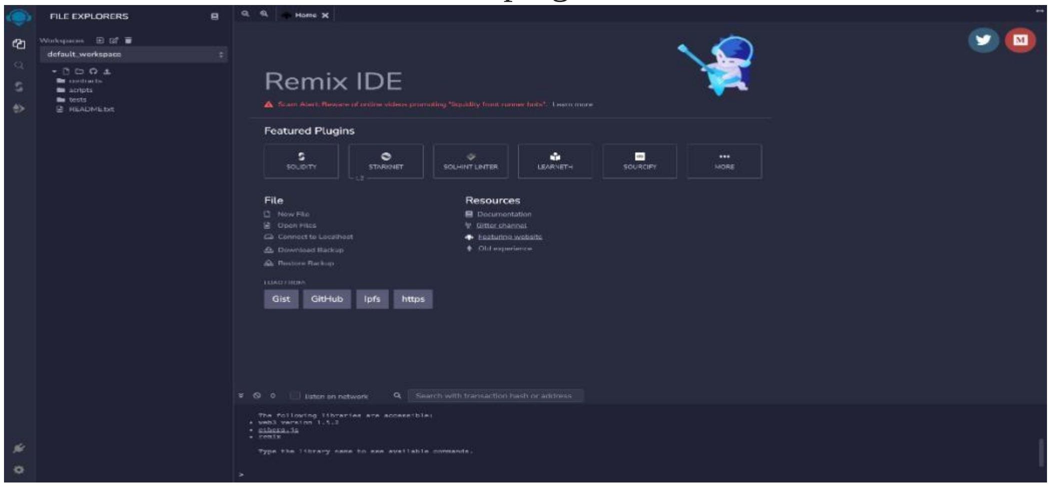
PRACTICAL NO: 04

AIM: Implement and demonstrate the use of following in solidity:

1. Variables
2. Operators
3. Loops
4. Decision making

NOTE:(Solidity is high level programming language and statically typed High level- humans can read and write in easy way & statically typed)
THEORY:

Remix ide

- Remix IDE, is a no-setup tool with a GUI for developing smart contracts. Used by experts and beginners alike, Remix will get you going in double time. Remix plays well with other tools, and allows for a simple deployment process to the chain of your choice. Remix is famous for our visual debugger. Remix is the place everyone comes to learn Ethereum.
- Remix Project is a platform for development tools that use a plugin architecture. It encompasses sub-projects including Remix Plugin Engine, Remix Libraries, and of course Remix IDE.
- Remix IDE is an open-source web and desktop application. It fosters a fast development cycle and has a rich set of plugins with intuitive GUIs. Remix is used for the entire journey of contract development with Solidity language as well as a playground for learning and teaching Ethereum.
- Start developing using Remix on browser, visit: https://remix.ethereum.org
- For desktop version, see releases: https://github.com/ethereum/remix-desktop/releases Remix libraries work as a core of native plugins of Remix IDE. Read more about libraries here
- 

```
The following libraries are accessible:
+ web3 version 1.5.2
+ ethers.js
+ remix

Type the library name to see available commands.
>
```

Solidity programming language

- Solidity is an object-oriented programming language for implementing smart contracts on various blockchain platforms, most notably, Ethereum. It was developed by Christian Reitwiessner, Alex Beregszaszi, and several former Ethereum core contributors. Programs in Solidity run on Ethereum Virtual Machine.
- Solidity is a statically typed programming language designed for developing smart contracts that run on the Ethereum Virtual Machine (EVM).
- Solidity uses ECMAScript-like syntax which makes it familiar for existing web developers; however unlike ECMAScript it has static typing and variadic return types. Solidity is different from other EVM-targeting languages such as Serpent and Mutan in some important ways. It supports complex member variables for contracts, including arbitrarily hierarchical mappings and structs. Solidity contracts support inheritance, including multiple inheritance with C3 linearization. Solidity introduces an application binary interface (ABI) that facilitates multiple type-safe functions within a single contract (this was also later supported by Serpent). The Solidity proposal also includes "Natural Language Specification", a documentation system for specifying user-centric descriptions of the ramifications of method-calls.
- Some key features of solidity are listed below:
  - Solidity is a high-level programming language designed for implementing smart contracts.
  - It is statically-typed object-oriented(contract-oriented) language. ○ Solidity is highly influenced by Python, c++, and JavaScript which runs on the Ethereum Virtual Machine(EVM).
  - Solidity supports complex user-defined programming, libraries and inheritance. ○ Solidity is primary language for blockchains running platforms. ○ Solidity can be used to creating contracts like voting, blind auctions, crowdfunding, multi-signature wallets, etc.
- DIFFERENT TYPES OF VARIABLES SCOPES IN SOLIDITY:
  1. State
  2. Local
  3. Global

State Variables − Variables whose values are permanently stored in a contract storage. Local Variables − Variables whose values are present till function is executing.

Global Variables − Special variables exists in the global namespace used to get information about the blockchain.

- DIFFERENT TYPES OF OPERATORS:

In any programming language, operators play a vital role i.e. they create a foundation for the programming. Similarly, the functionality of Solidity is also incomplete without the use of operators. Operators allow users to perform different operations on operands. Solidity supports the following types of operators based upon their functionality.

○ Arithmetic Operators
○ Relational Operators
○ Logical Operators ○
Conditional Operator

ARITHEMATIC OPERATORS:

Operator	Denotation	Description	Syntax
Addition	+	Used to add two operands	uint c=a+b;
Subtraction	–	Used to subtract the second operand from first	uint c=a-b;
Multiplication	*	Used to multiply both operands	uint c=a*b;
Division	/	Used to divide numerator by denominator	uint c=a/b;
Modulus	%	Gives the remainder after integer division	uint c=a%b;
Increment	++	Increases the integer value by one	uint c=++a;
Decrement	—	Decreases the integer value by one	uint c=--b;

RELATIONAL OPERATORS:

Operator	Denotation	Description	Syntax
Equal	==	Checks if two values are equal or not, returns true if equals, and vice-versa	Bool c=a==b;
Not Equal	!=	Checks if two values are equal or not, returns true if not equals, and vice-versa	Bool c=a! =b;
Greater than	>	Checks if left value is greater than right or not, returns true if greater, and vice-versa	Bool c=a>b;
Less than	<	Checks if left value is less than right or not, returns true if less, and vice-versa	Bool c=a<b;
Greater than or Equal to	>=	Checks if left value is greater and equal than right or not, returns true if greater and equal, and vice-versa	Bool c=a>=b;
Less than or Equal to	<=	Checks if left value is less than right or not, returns true if less and equals, and vice-versa	Bool c=a<=b;

LOGICAL OPERATORS:

Operator	Denotation	Description	Syntax
Logical AND	&&	Returns true if both conditions are true and false if one or both conditions are false	Bool c=bool a && bool b;

Logical OR	||	Returns true if one or both conditions are true and false when both are false	Bool c=bool a || bool b;
Logical NOT	!	Returns true if the condition is not satisfied else false	Bool c=! bool a;

CONDITIONAL OPERATOR:
It is a ternary operator that evaluates the expression first then checks the condition for return values corresponding to true or false.
Syntax: if condition true ? then
A: else B

LOOPS IN SOLIDITY
1. While loop
2. For loop
3. Do while loop

WHILE LOOP:
This is the most basic loop in solidity, Its purpose is to execute a statement or block of statements repeatedly as far as the condition is true and once the condition becomes false the loop terminates.
Syntax:
while (condition) {
statement or block of code to be executed if the condition is True }

FOR LOOP:
This is the most compact way of looping. It takes three arguments separated by a semi-colon to run. The first one is 'loop initialization' where the iterator is initialized with starting value, this statement is executed before the loop starts. Second is 'test statement' which checks whether the condition is true or not, if the condition is true the loop executes else terminates. The third one is the 'iteration statement' where the iterator is increased or decreased. Below is the syntax of for loop:
Syntax:
for (initialization; test condition; iteration statement) {
statement or block of code to be executed if the condition is True }

DO WHILE LOOP:

This loop is very similar to while loop except that there is a condition check which happens at the end of loop i.e. the loop will always execute at least one time even if the condition is false.

Syntax:

```
do  {
   block of statements to be executed
} while (condition);
```

DECISION MAKING STATEMENTS:

1. IF statement
2. IF ELSE statement
3. IF ELSEIF ELSE statement

If statement

This is the most basic conditional statement. It is used to make a decision whether the statement or block of code will be executed or not. If the condition is true the statements will be executed, else no statement will execute.

Syntax:

```
if (condition) {
   statement or block of code to be executed if the condition is True }
```

if…else statement

This statement is the next form of conditional statement which allows the program to execute in a more controlled way. Here if the condition is true then the, if block is executed while if the condition is false then else block, is executed.

Syntax:

```
if (condition) {
   statement or block of code to be executed if condition is True
} else {
   statement or block of code to be executed if condition is False }
```

if…else if…else statement

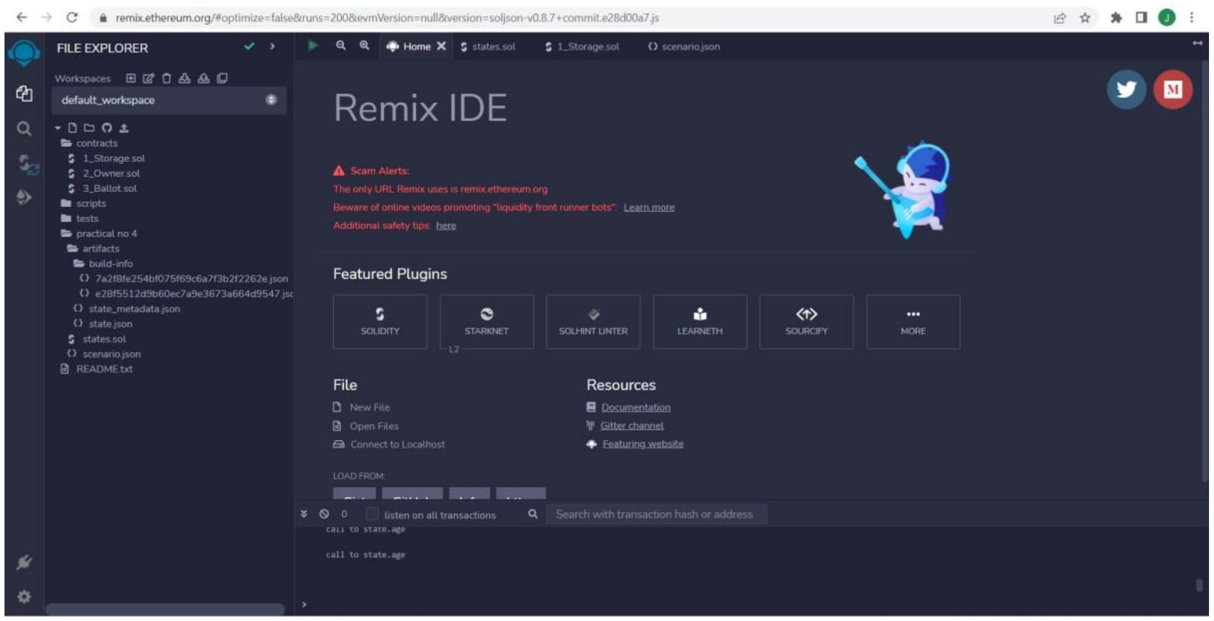
This is a modified form of if…else conditional statement which is used to make a decision among several options. The statements start execution from if statement and as the condition of any if block is true the block of code associated with it is executed and rest if are skipped, and if none of the condition is true then else block executes.

Syntax:

```
if (condition) {
   statement or block of code to be executed if the condition is True
} else if (condition 2) {
   statement or block of code to be executed if the condition of else...if is True
} else {
   statement or block of code to be executed if none of the condition is True }
```

Steps to run solidity program

- STEP1: Open remix IDE https://remix-project.org/
  scroll down and click on remix online IDE
  the below screen will be visible which is home screen of remix IDE
  here click on folder icon and create a new folder named"practical no 4" and inside this folder create new file named states.



remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.7+commit.e28d00a7.js

Scam Alerts:
The only URL Remix uses is remix.ethereum.org
Beware of online videos promoting "liquidity front runner bots": Learn more
Additional safety tips: here

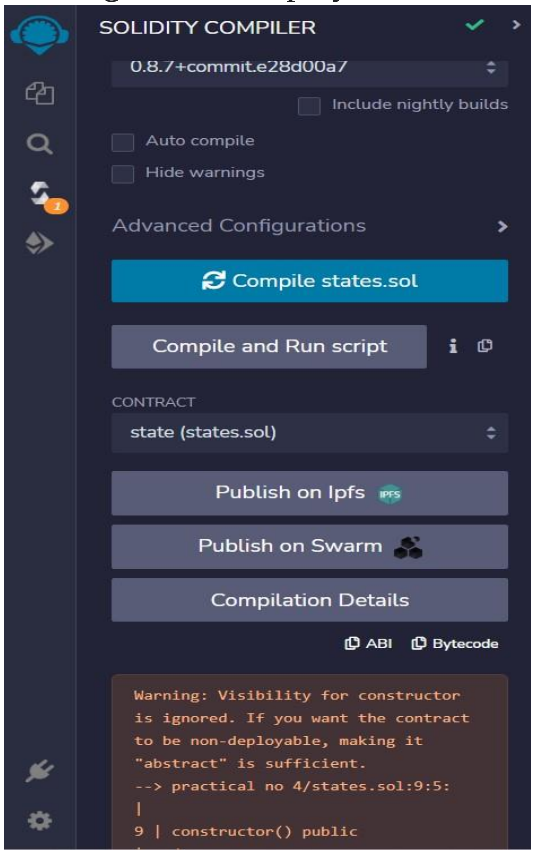
```
call to state.age
call to state.age
```

- STEP 2: click on the stats file and start typing the code:
  First 2 lines are important and will be same for all codes which will be written.

```
// SPDX-License-Identifier: GPL-3.0
 pragma solidity >=0.7.0
<0.9.0;
 contract
state{
     uint public
age;
         constructor()
public
    {
        age=33;
    }
}
```

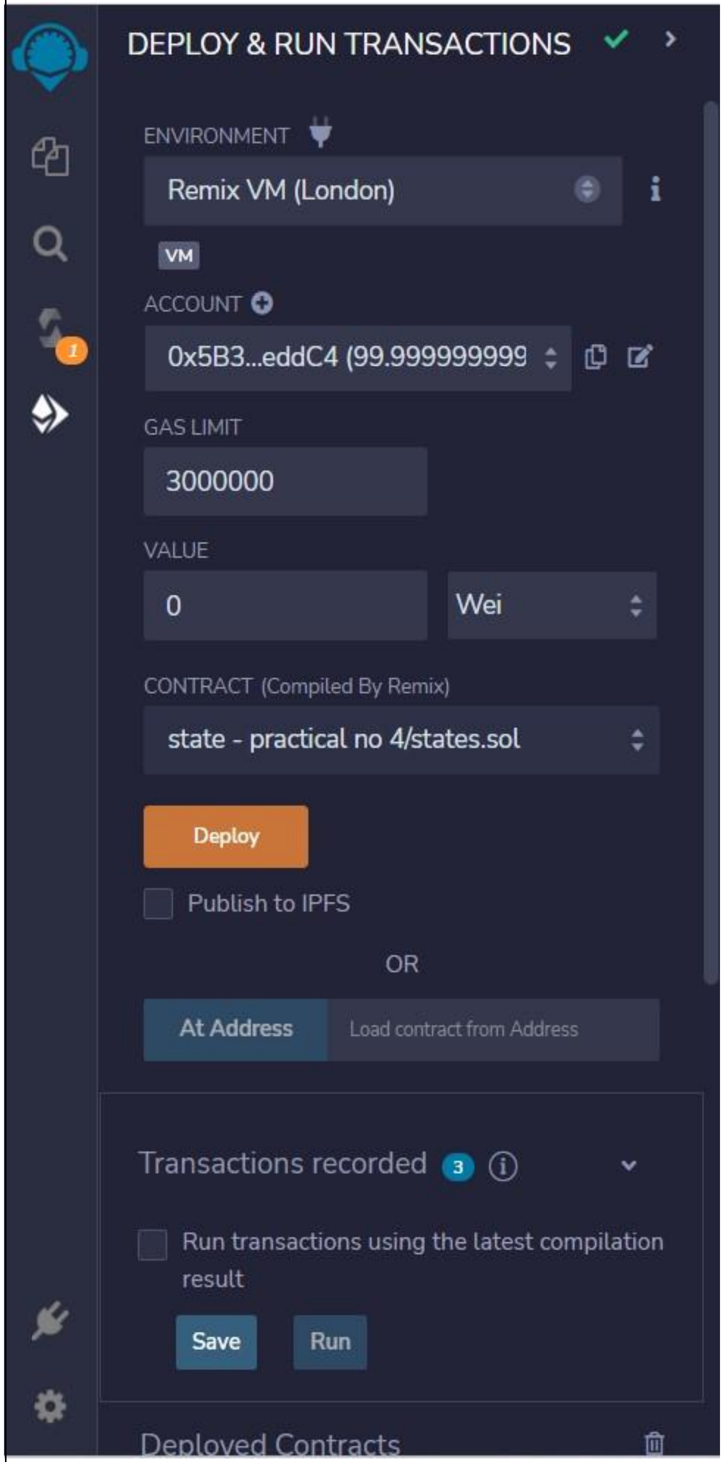
- STEP 3: Once the code is written click on the compile button which is a play button (green coloured) on top left corner or on the left below the search option there is a compile button click on it.

NOTE: in order to compile the code first you have to be on the file which you need compile and then on the compile button the name of file will be shown and then hit on compile. It it shows green tick then program is successfully compiled and has no error however in some case the warning will be displayed that can be ignored.

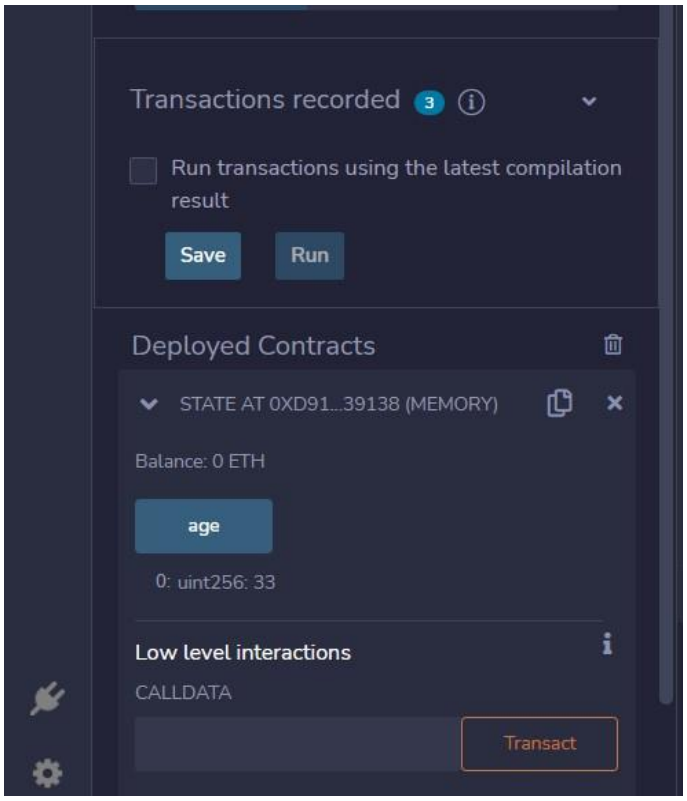


```
Warning: Visibility for constructor
is ignored. If you want the contract
to be non-deployable, making it
"abstract" is sufficient.
--> practical no 4/states.sol:9:5:
|
9 | constructor() public
```

- STEP 4: Deploy the program
  Once compiled successfully click on the deploy button just below the compile button and then click on deploy
  Don't change any default setting
- STEP 5: Once deployed scroll down on the same menu there you can see the deployed contract with a button age. When you click on the button it will show the output


DEPLOY & RUN TRANSACTIONS

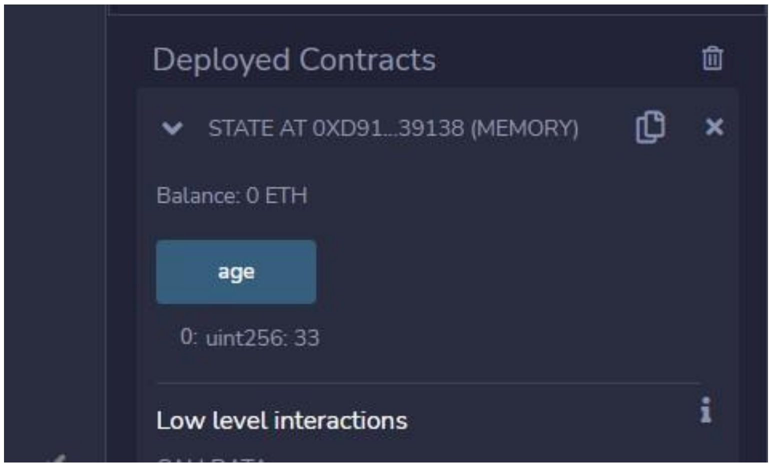
ENVIRONMENT
Remix VM (London)
VM
ACCOUNT
0x5B3...eddC4 (99.999999999
GAS LIMIT
3000000
VALUE
0 Wei
CONTRACT (Compiled By Remix)
state - practical no 4/states.sol
Deploy
Publish to IPFS
OR
At Address | Load contract from Address
Transactions recorded 3
Run transactions using the latest compilation result
Save Run
Deployed Contracts


Transactions recorded 3
Run transactions using the latest compilation result
Save Run
Deployed Contracts
STATE AT 0XD91...39138 (MEMORY)
Balance: 0 ETH
age
0: uint256: 33
Low level interactions
CALLDATA
Transact

- OUTPUT:

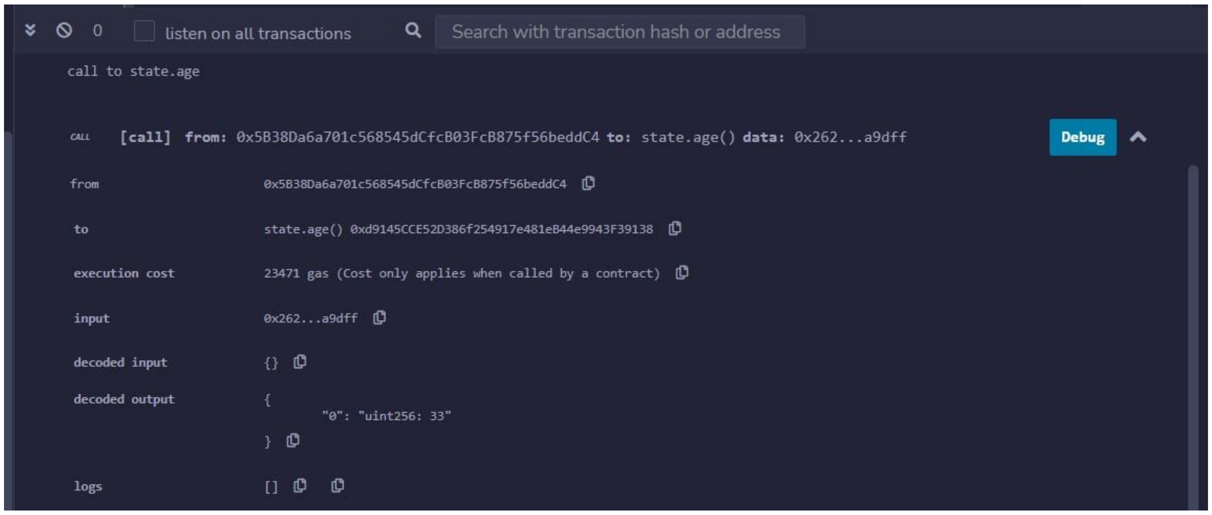
  There are 2 ways to check output:

1. Below the deployed contracts you can see the output



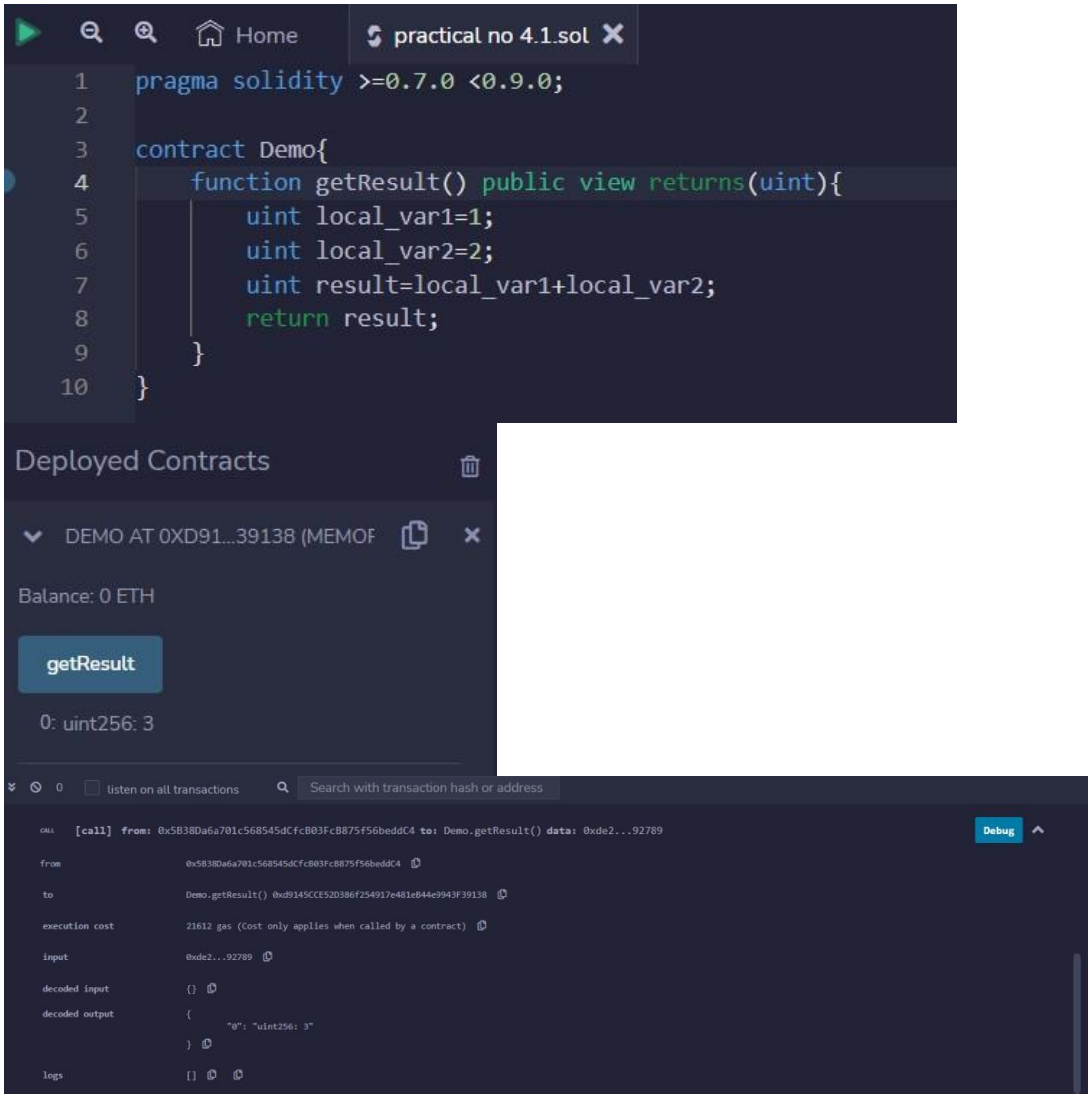
```
Balance: 0 ETH
0: uint256: 33
```

2. Below the code which you have written there is a log tab when you hit on age button it generates some log data in that there is a debug button so you can click on it you will get a detailed output.


listen on all transactions
Search with transaction hash or address
Debug

```
call to state.age

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: state.age() data: 0x262...a9dff

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                state.age() 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost    23471 gas (Cost only applies when called by a contract)
input             0x262...a9dff
decoded input     {}
decoded output    {
                          "0": "uint256: 33"
                  }
logs              []
```

LOCAL VARIABLES:



Home
practical no 4.1.sol

```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    function getResult() public view returns(uint){
        uint local_var1=1;
        uint local_var2=2;
        uint result=local_var1+local_var2;
        return result;
    }
}
```

Deployed Contracts

DEMO AT 0XD91...39138 (MEMOF

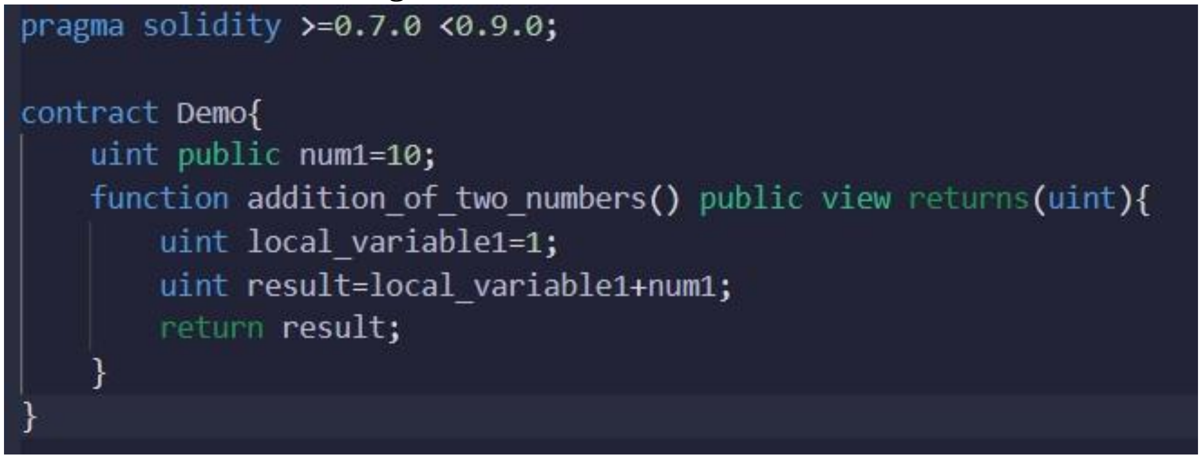
Balance: 0 ETH

```
0: uint256: 3
```

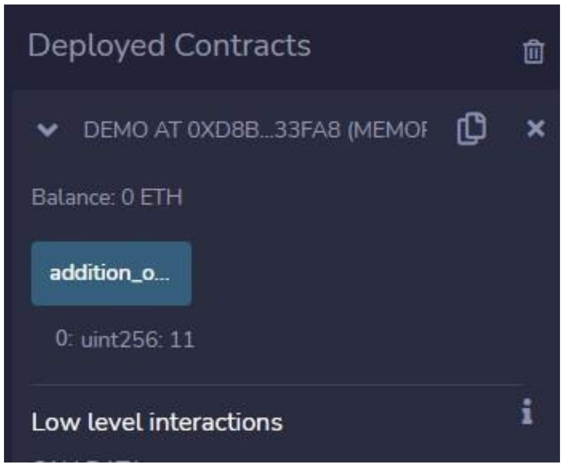
0
listen on all transactions
Search with transaction hash or address
Debug

```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Demo.getResult() data: 0xde2...92789

from              0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                Demo.getResult() 0xd9145CCE52D386f254917e481eB44e9943F39138
execution cost    21612 gas (Cost only applies when called by a contract)
input             0xde2...92789
decoded input     {}
decoded output    {
                        "0": "uint256: 3"
                  }
logs              []
```

Local and state variable together:



```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    uint public num1=10;
    function addition_of_two_numbers() public view returns(uint){
        uint local_variable1=1;
        uint result=local_variable1+num1;
        return result;
    }
}
```



Deployed Contracts

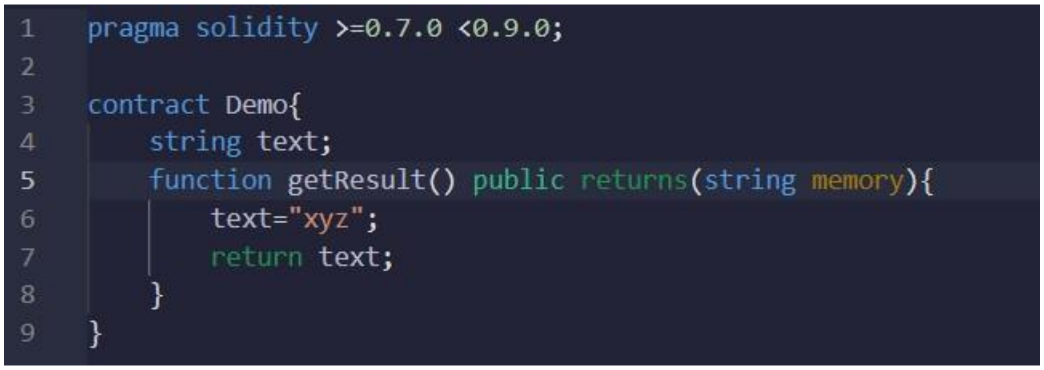
DEMO AT 0XD8B...33FA8 (MEMOI

Balance: 0 ETH

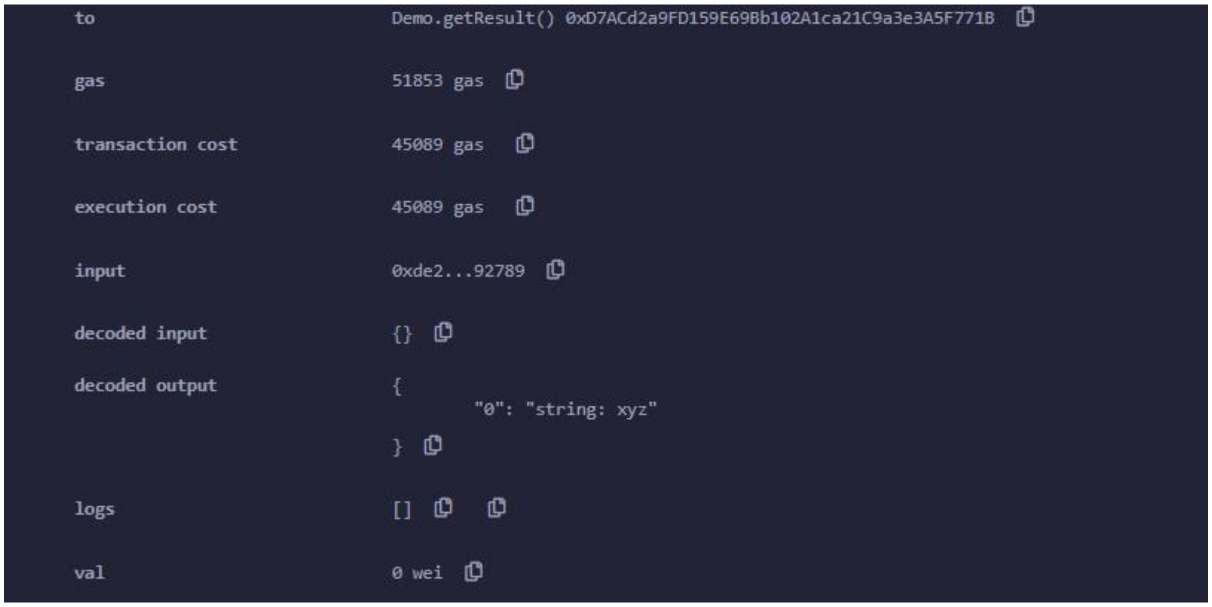
```
0: uint256: 11
```

Low level interactions

STRING VARIABLES

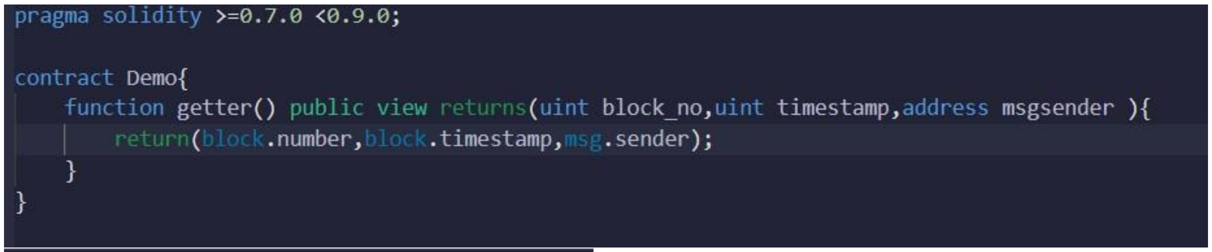


```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    string text;
    function getResult() public returns(string memory){
        text="xyz";
        return text;
    }
}
```

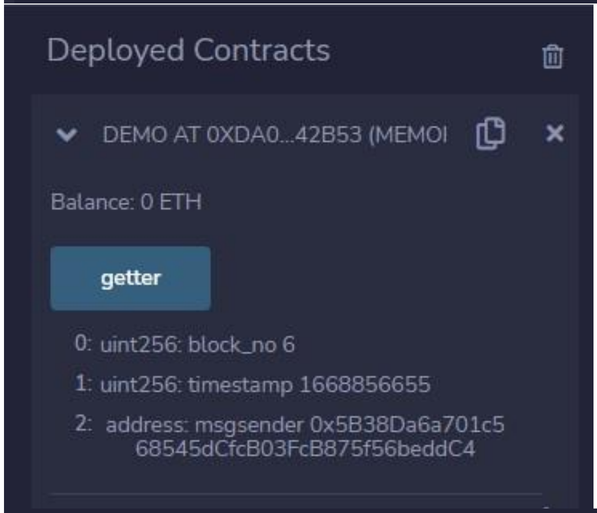


```
to                  Demo.getResult() 0xD7ACd2a9FD159E69Bb102A1ca21C9a3e3A5F771B

gas                 51853 gas

transaction cost    45089 gas

execution cost      45089 gas

input               0xde2...92789

decoded input       {}

decoded output      {
                            "0": "string: xyz"
                    }

logs                []

val                 0 wei
```

GLOBAL VARIABLES:



```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    function getter() public view returns(uint block_no,uint timestamp,address msgsender ){
        return(block.number,block.timestamp,msg.sender);
    }
}
```

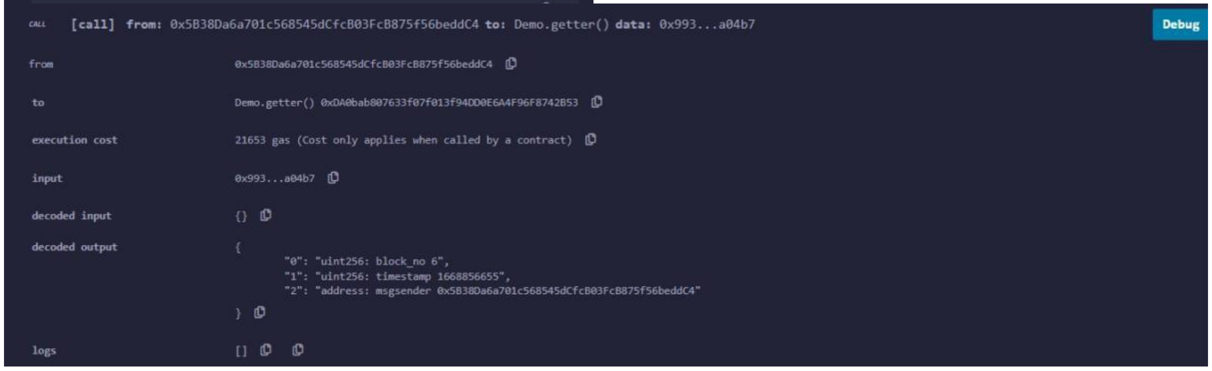


Deployed Contracts

DEMO AT 0XDA0...42B53 (MEMOI

Balance: 0 ETH

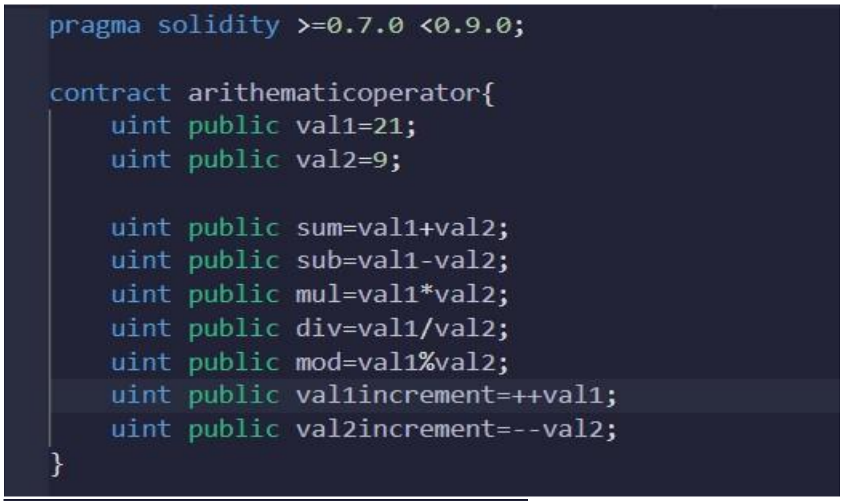
```
0: uint256: block_no 6
1: uint256: timestamp 1668856655
2: address: msgsender 0x5B38Da6a701c5
   68545dCfcB03FcB875f56beddC4
```



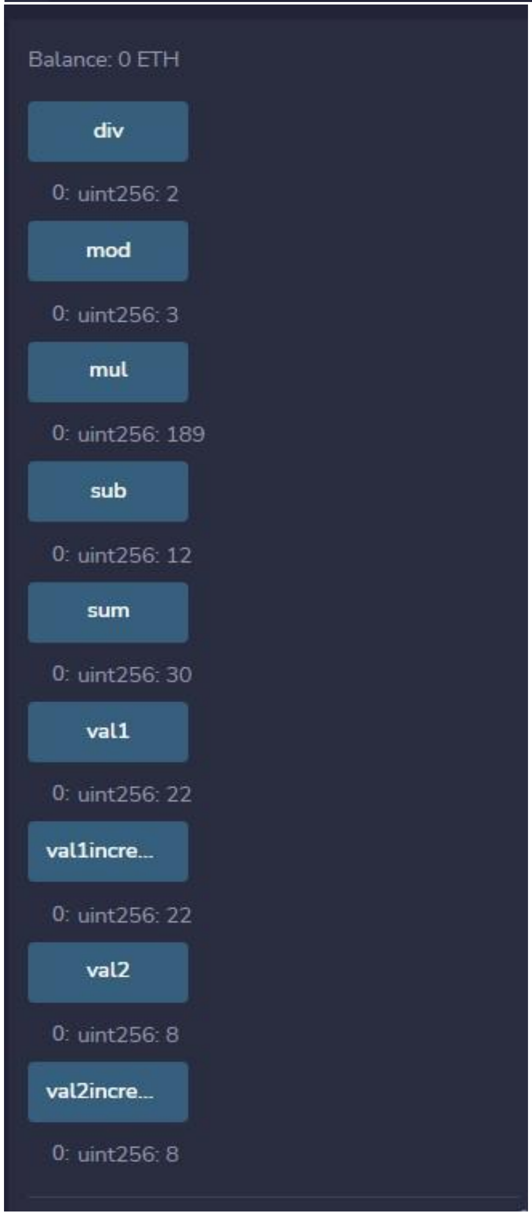
```
CALL    [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Demo.getter() data: 0x993...a04b7

from                  0x5B38Da6a701c568545dCfcB03FcB875f56beddC4

to                    Demo.getter() 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53

execution cost        21653 gas (Cost only applies when called by a contract)

input                 0x993...a04b7

decoded input         {}

decoded output        {
                              "0": "uint256: block_no 6",
                              "1": "uint256: timestamp 1668856655",
                              "2": "address: msgsender 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"
                      }

logs                  []
```

OPERATORS

ARITHEMATIC OPERATORS:

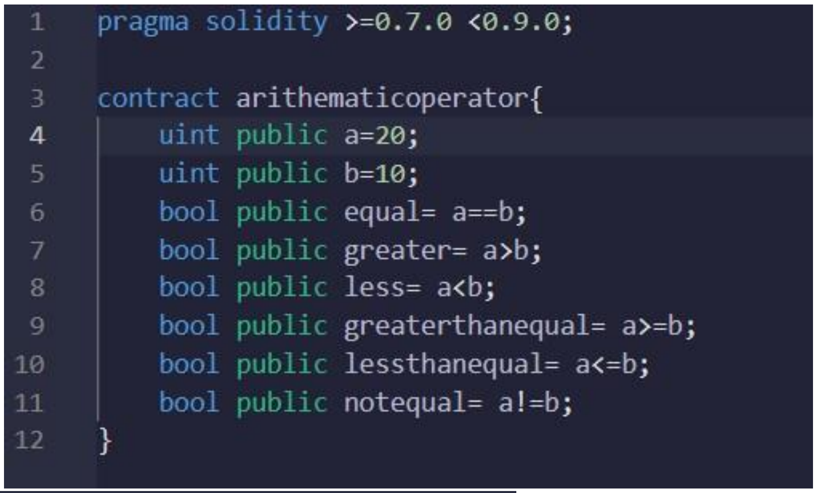


```
pragma solidity >=0.7.0 <0.9.0;

contract arithematicoperator{
    uint public val1=21;
    uint public val2=9;

    uint public sum=val1+val2;
    uint public sub=val1-val2;
    uint public mul=val1*val2;
    uint public div=val1/val2;
    uint public mod=val1%val2;
    uint public val1increment=++val1;
    uint public val2increment=--val2;
}
```

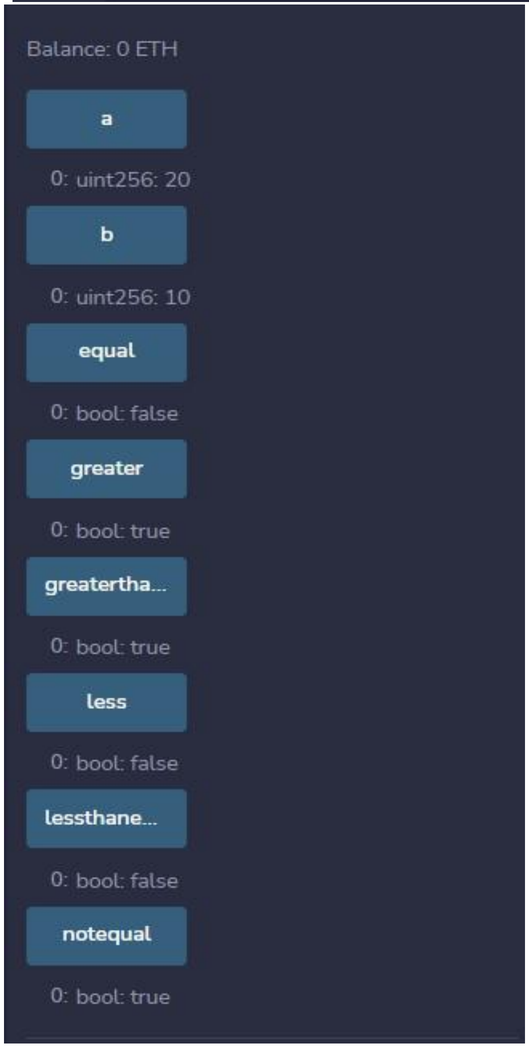


```
Balance: 0 ETH
div
0: uint256: 2
mod
0: uint256: 3
mul
0: uint256: 189
sub
0: uint256: 12
sum
0: uint256: 30
val1
0: uint256: 22
val1incre...
0: uint256: 22
val2
0: uint256: 8
val2incre...
0: uint256: 8
```

RELATIONAL OPERATORS:

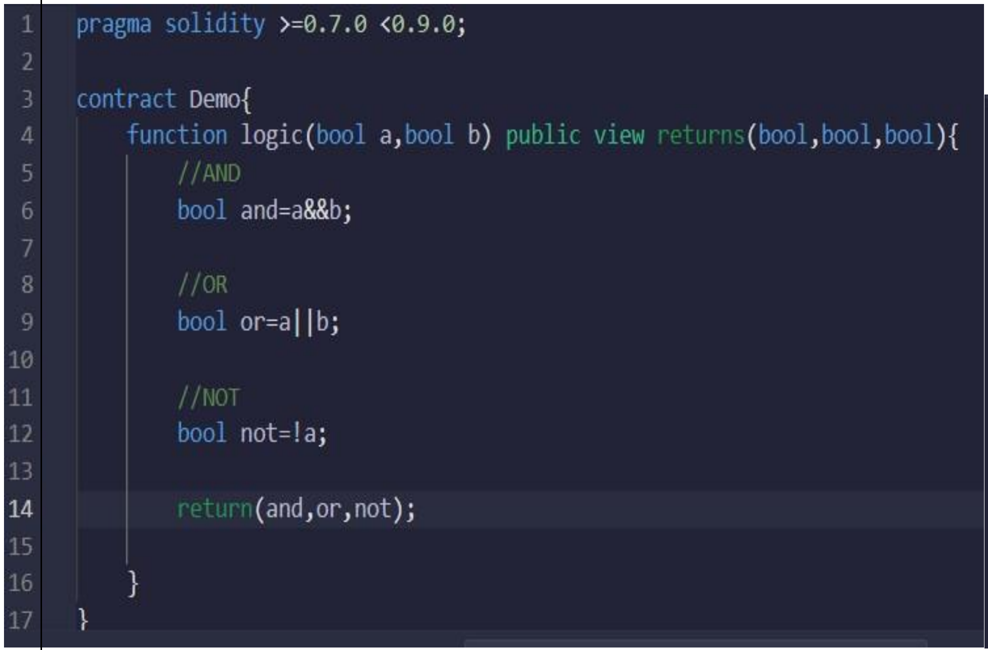


```
pragma solidity >=0.7.0 <0.9.0;

contract arithematicoperator{
    uint public a=20;
    uint public b=10;
    bool public equal= a==b;
    bool public greater= a>b;
    bool public less= a<b;
    bool public greaterthanequal= a>=b;
    bool public lessthanequal= a<=b;
    bool public notequal= a!=b;
}
```

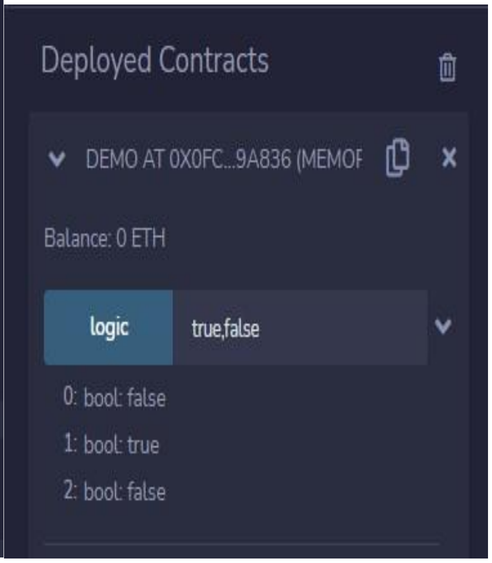


```
Balance: 0 ETH
a
0: uint256: 20
b
0: uint256: 10
equal
0: bool: false
greater
0: bool: true
greatertha...
0: bool: true
less
0: bool: false
lessthane...
0: bool: false
notequal
0: bool: true
```

LOGICAL OPERATORS:

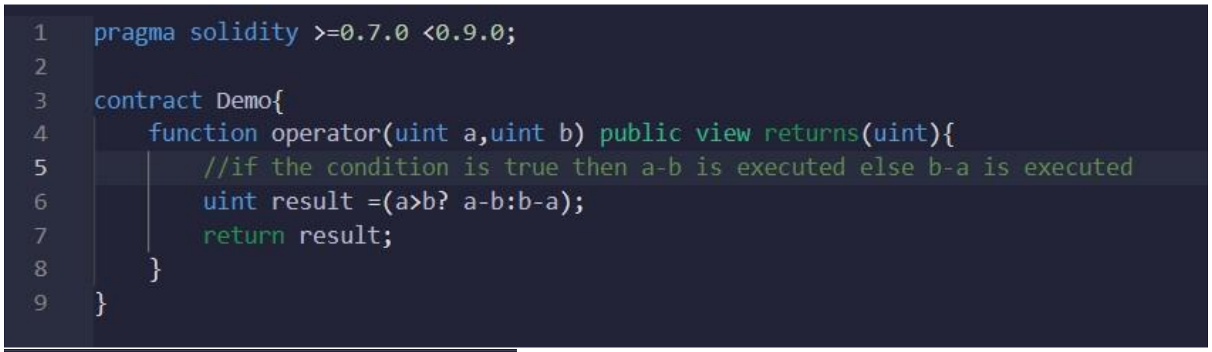


```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    function logic(bool a,bool b) public view returns(bool,bool,bool){
        //AND
        bool and=a&&b;

        //OR
        bool or=a||b;

        //NOT
        bool not=!a;

        return(and,or,not);

    }
}
```

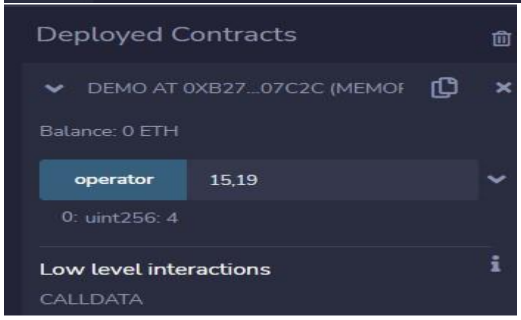


```
Deployed Contracts
DEMO AT 0X0FC...9A836 (MEMOF
Balance: 0 ETH
logic    true,false
0: bool: false
1: bool: true
2: bool: false
```

CONDITIONAL OPERATOR:



```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    function operator(uint a,uint b) public view returns(uint){
        //if the condition is true then a-b is executed else b-a is executed
        uint result =(a>b? a-b:b-a);
        return result;
    }
}
```

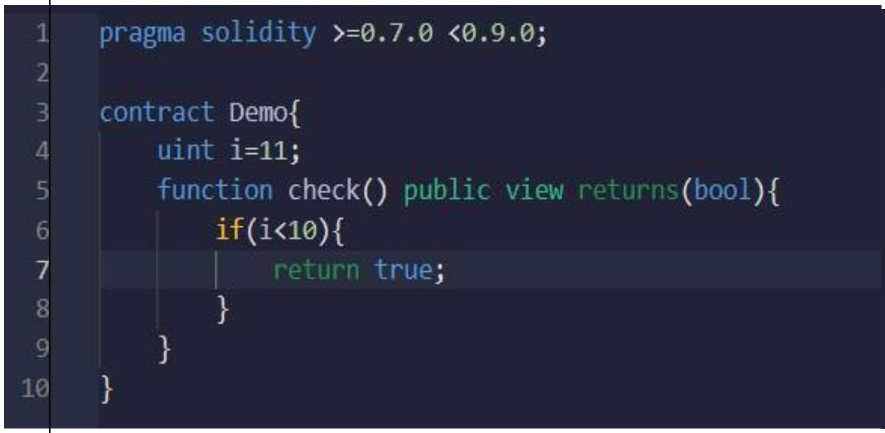


```
Deployed Contracts
DEMO AT 0XB27...07C2C (MEMOF
Balance: 0 ETH
operator    15,19
0: uint256: 4
Low level interactions
CALLDATA
```

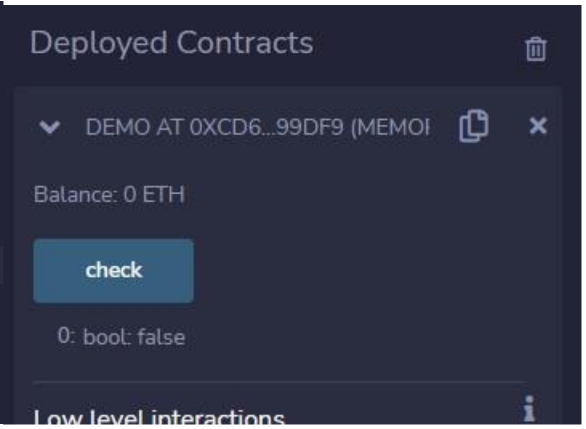
OTHER CONDITIONAL OPERATOR EXAMPLES:

```
//bool result=((a*b)<c? true:false);
```

DECISION MAKING PART 3:
IF STATEMENT

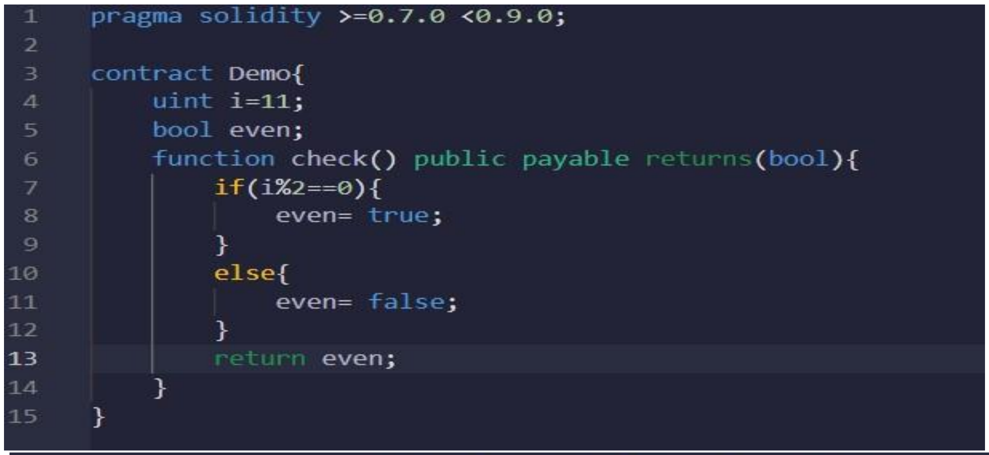


```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    uint i=11;
    function check() public view returns(bool){
        if(i<10){
            return true;
        }
    }
}
```

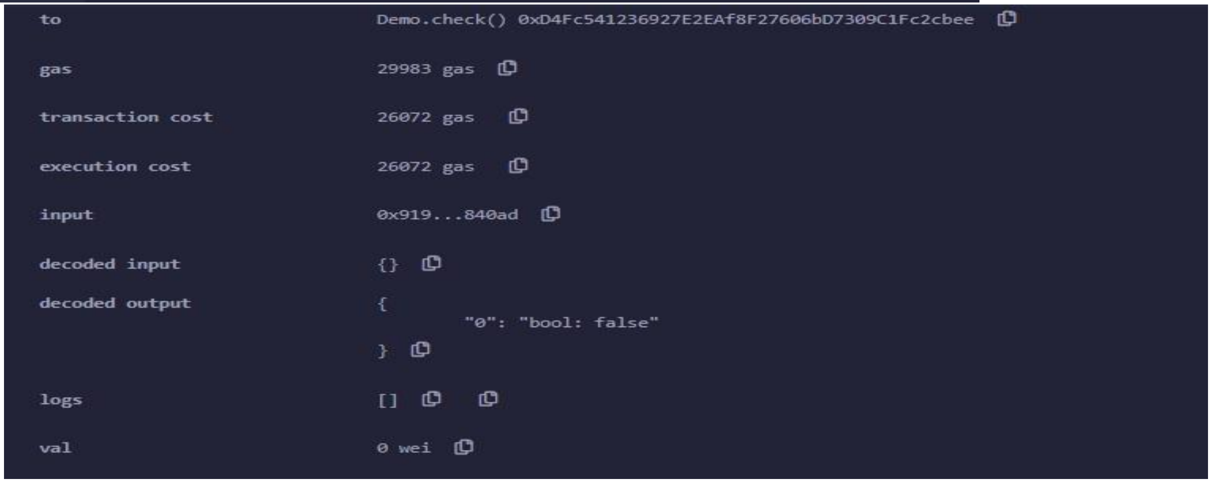


```
Deployed Contracts
DEMO AT 0XCD6...99DF9 (MEMOI
Balance: 0 ETH
check
0: bool: false
Low level interactions
```

IF ELSE STATEMENT:

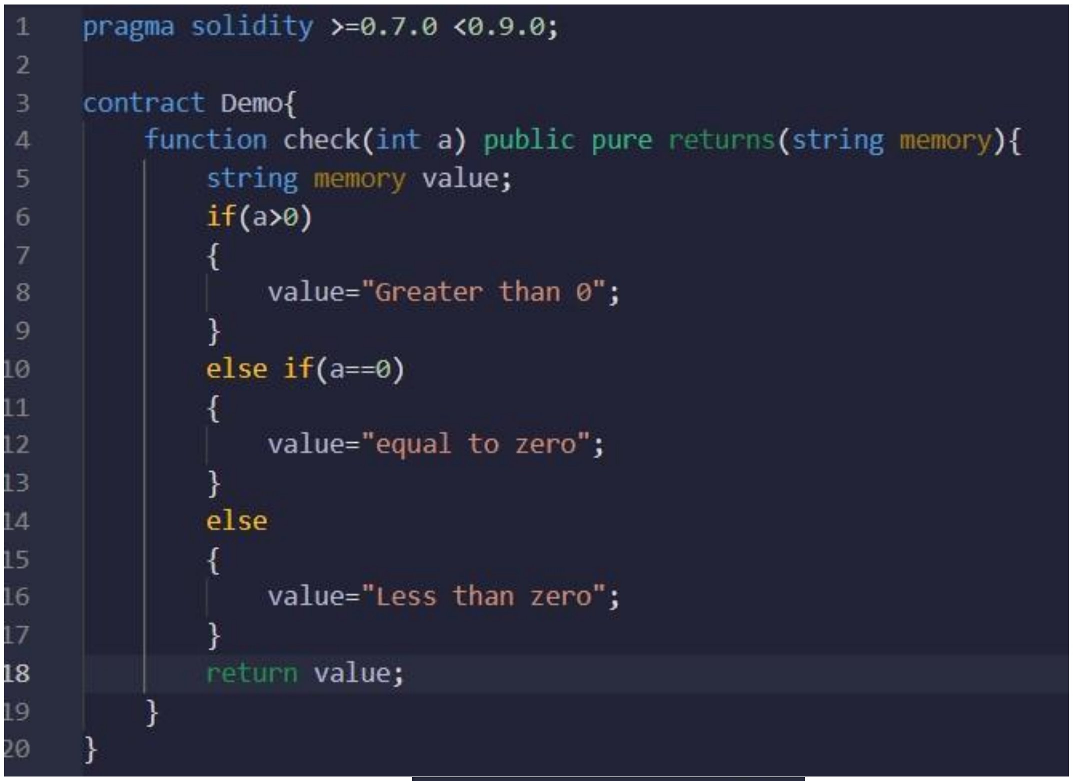


```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    uint i=11;
    bool even;
    function check() public payable returns(bool){
        if(i%2==0){
            even= true;
        }
        else{
            even= false;
        }
        return even;
    }
}
```

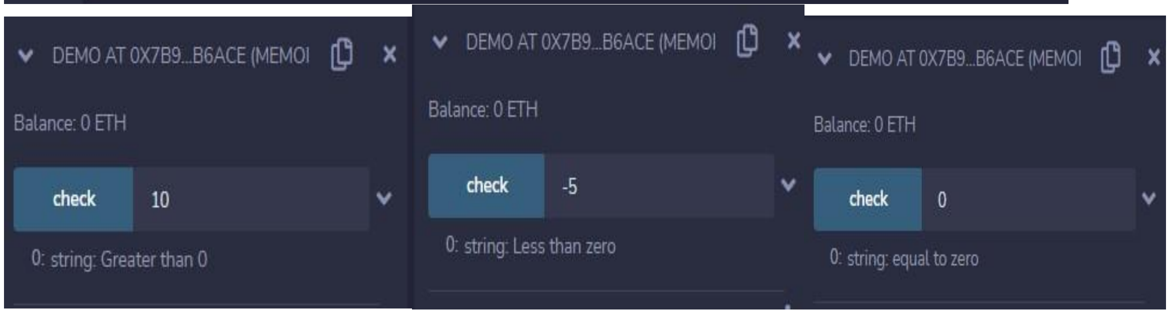


```
to                      Demo.check() 0xD4Fc541236927E2EAf8F27606bD7309C1Fc2cbee
gas                     29983 gas
transaction cost        26072 gas
execution cost          26072 gas
input                   0x919...840ad
decoded input           {}
decoded output          {
                            "0": "bool: false"
                        }
logs                    []
val                     0 wei
```

IF ELSE IF STATEMENT



```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    function check(int a) public pure returns(string memory){
        string memory value;
        if(a>0)
        {
            value="Greater than 0";
        }
        else if(a==0)
        {
            value="equal to zero";
        }
        else
        {
            value="Less than zero";
        }
        return value;
    }
}
```

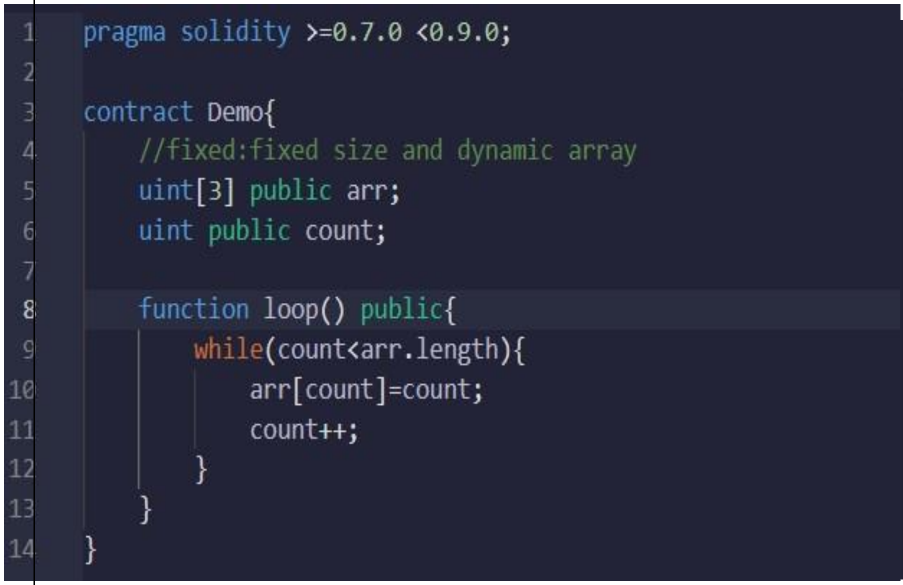


```
DEMO AT 0X7B9...B6ACE (MEMOI
Balance: 0 ETH
check 10
0: string: Greater than 0
```

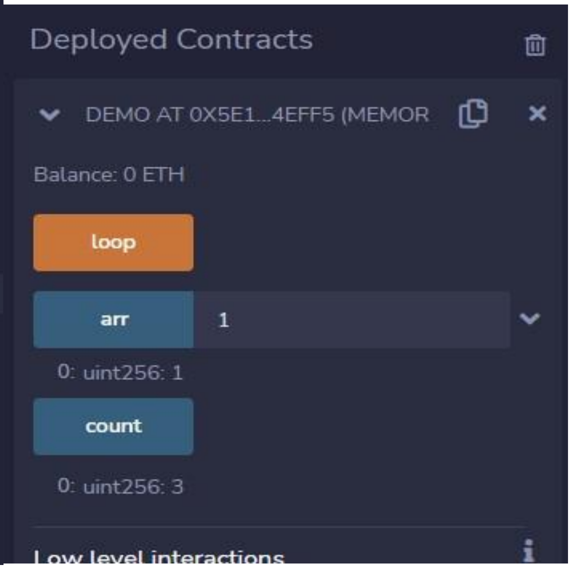
```
DEMO AT 0X7B9...B6ACE (MEMOI
Balance: 0 ETH
check -5
0: string: Less than zero
```

```
DEMO AT 0X7B9...B6ACE (MEMOI
Balance: 0 ETH
check 0
0: string: equal to zero
```

LOOPING STATEMENT:
WHILE LOOP



```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    //fixed:fixed size and dynamic array
    uint[3] public arr;
    uint public count;

    function loop() public{
        while(count<arr.length){
            arr[count]=count;
            count++;
        }
    }
}
```



Deployed Contracts

DEMO AT 0X5E1...4EFF5 (MEMOR

Balance: 0 ETH

loop

arr 1

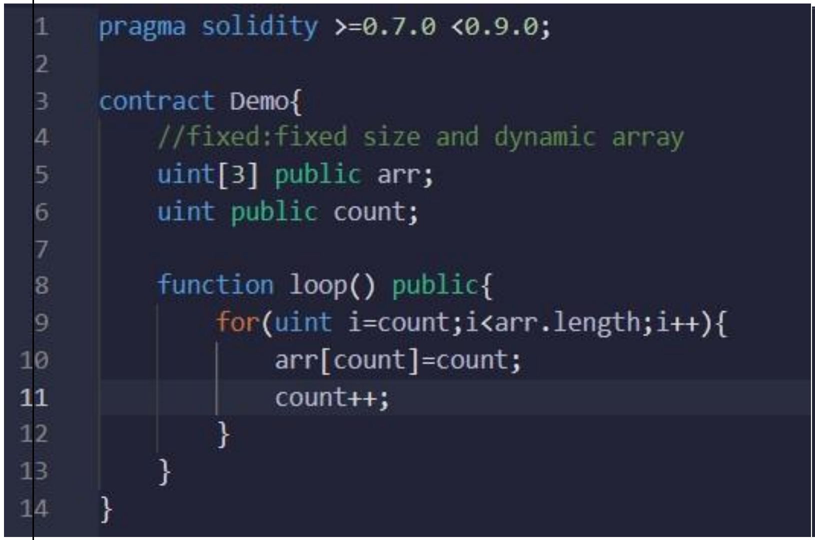
0: uint256: 1

count

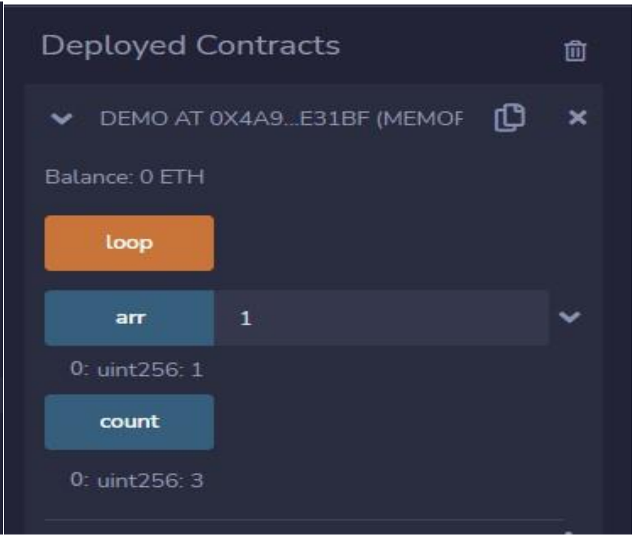
0: uint256: 3

Low level interactions

FOR LOOP:



```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    //fixed:fixed size and dynamic array
    uint[3] public arr;
    uint public count;

    function loop() public{
        for(uint i=count;i<arr.length;i++){
            arr[count]=count;
            count++;
        }
    }
}
```



Deployed Contracts

DEMO AT 0X4A9...E31BF (MEMOF

Balance: 0 ETH

loop

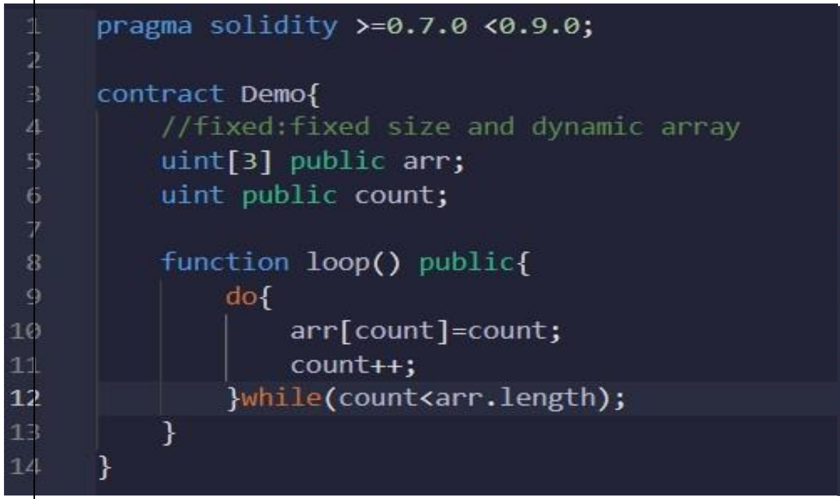
arr 1

0: uint256: 1

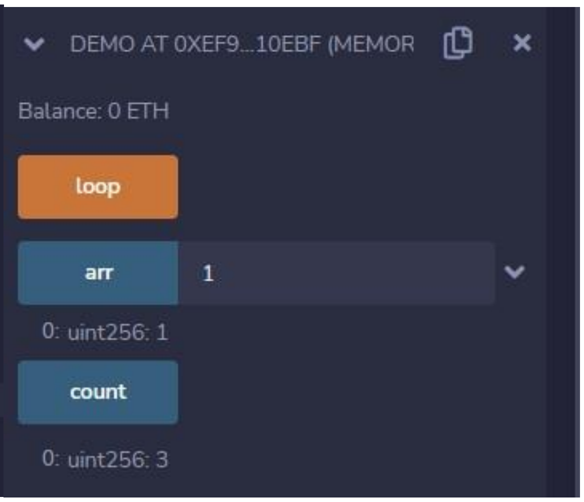
count

0: uint256: 3

DO WHILE LOOP



```
pragma solidity >=0.7.0 <0.9.0;

contract Demo{
    //fixed:fixed size and dynamic array
    uint[3] public arr;
    uint public count;

    function loop() public{
        do{
            arr[count]=count;
            count++;
        }while(count<arr.length);
    }
}
```



DEMO AT 0XEF9...10EBF (MEMOR

Balance: 0 ETH

loop

arr 1

0: uint256: 1

count

0: uint256: 3

PRACTICAL NO: 05

AIM: Implement and demonstrate the use of following in solidity:
1. Strings
2. Arrays
3. Enums
4. Structs
5. Mappings

1. STRINGS:

Strings in Solidity is a reference type of data type which stores the location of the data instead of directly storing the data into the variable.
They are dynamic arrays that store a set of characters that can consist of numbers, special characters, spaces, and alphabets.
Strings in solidity store data in UTF-8 encoding.
Like JavaScript, both Double quote (" ") and Single quote(' ') can be used to represent strings in solidity.
"Hello World" // Valid string
'Hello World' // Valid string
'Hello World" // Invalid string
SYNTAX:
String variable_name="text";

2. ARRAYS:

Arrays are data structures that store the fixed collection of elements of the same data types in which each and every element has a specific location called index. Instead of creating numerous individual variables of the same type, we just declare one array of the required size and store the elements in the array and can be accessed using the index. In Solidity, an array can be of fixed size or dynamic size.
To declare an array in Solidity, the data type of the elements and the number of elements should be specified. The size of the array must be a positive integer and data type should be a valid Solidity type Syntax:
<data type> <array name>[size] = <initialization>

3. ENUMS:

Enums are the way of creating user-defined data types, it is usually used to provide names for integral constants which makes the contract better for maintenance and reading. Enums restrict the variable with one of a few predefined values, these values of the enumerated list are called enums. Options are represented with integer values starting from zero, a default value can also be given for the enum. By using enums it is possible to reduce the bugs in the code. Syntax:

enum <enumerator_name> {
element 1, element 2,....,element n }

4. **STRUCTS:**

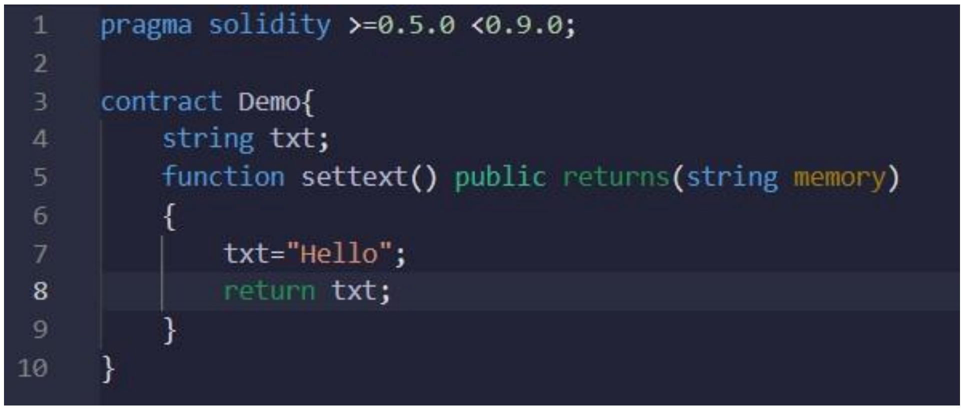
Structs in Solidity allows you to create more complicated data types that have multiple properties.
You can define your own type by creating a struct.
They are useful for grouping together related data.
Structs can be declared outside of a contract and imported in another contract. Generally, it is used to represent a record. To define a structure struct keyword is used, which creates a new data type.
Syntax: struct
<structure_name> {
<data type> variable_1;
<data type> variable_2; }

5. **MAPPING:**

Mapping in Solidity acts like a hash table or dictionary in any other language. These are used to store the data in the form of key-value pairs, a key can be any of the built-in data types but reference types are not allowed while the value can be of any type. Mappings are mostly used to associate the unique Ethereum address with the associated value type.
Syntax:
mapping (key => value) <access specifier> <name>;

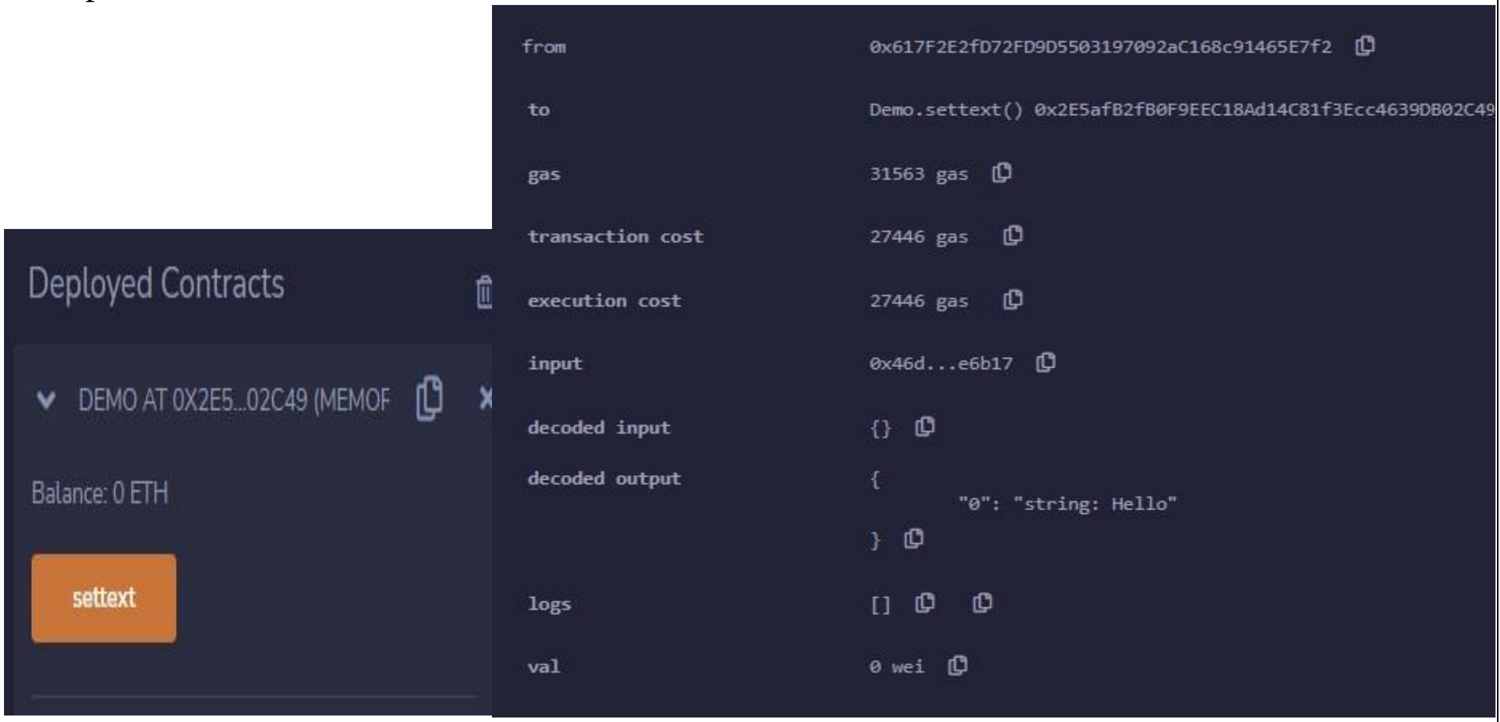
CODES:

1. Strings:



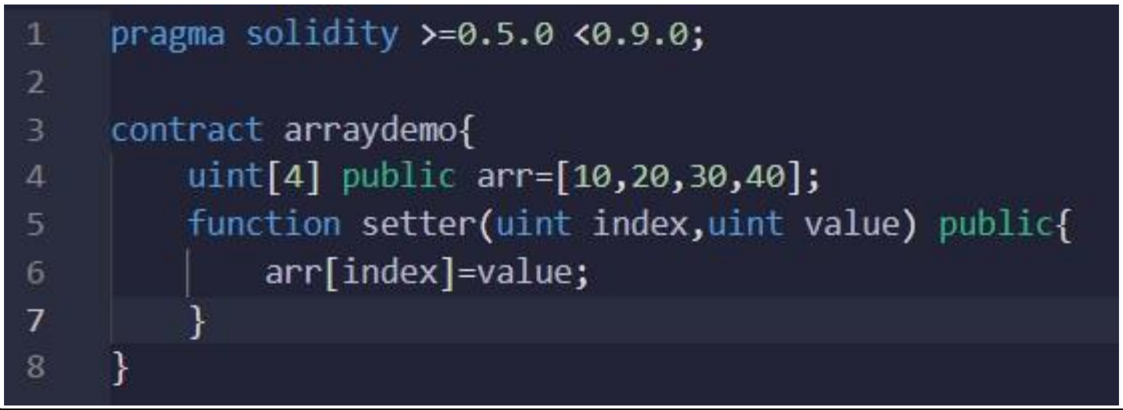
```
pragma solidity >=0.5.0 <0.9.0;

contract Demo{
    string txt;
    function settext() public returns(string memory)
    {
        txt="Hello";
        return txt;
    }
}
```

Output:



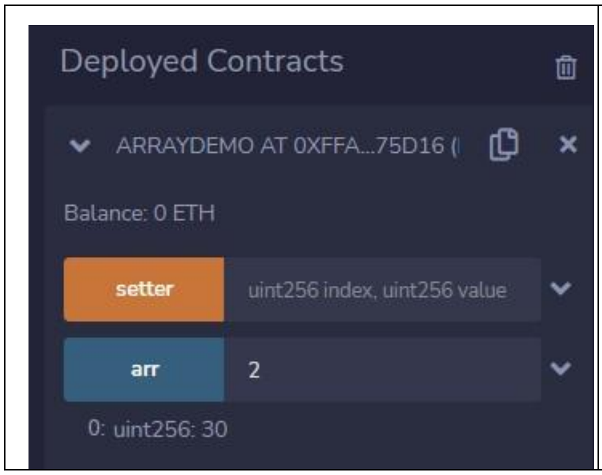
```
from                0x617F2E2fD72FD9D5503197092aC168c91465E7f2
to                  Demo.settext() 0x2E5afB2fB0F9EEC18Ad14C81f3Ecc4639DB02C49
gas                 31563 gas
transaction cost    27446 gas
execution cost      27446 gas
input               0x46d...e6b17
decoded input       {}
decoded output      {
                            "0": "string: Hello"
                    }
logs                []
val                 0 wei
```

2. ARRAYS;



```
pragma solidity >=0.5.0 <0.9.0;

contract arraydemo{
    uint[4] public arr=[10,20,30,40];
    function setter(uint index,uint value) public{
        arr[index]=value;
    }
}
```

BEFORE UPDATE	AFTER UPDATE



Deployed Contracts

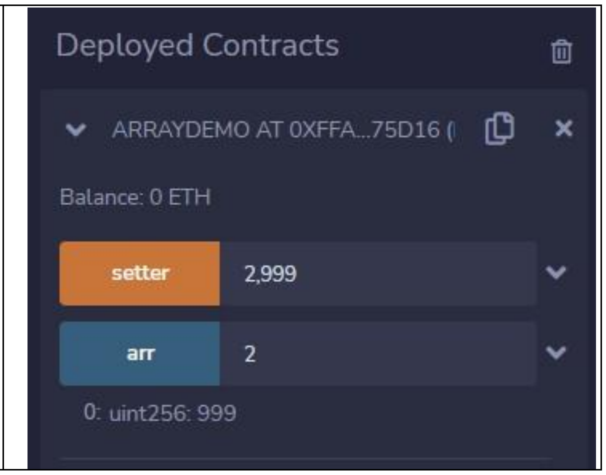
ARRAYDEMO AT 0XFFA...75D16 (

Balance: 0 ETH

setter uint256 index, uint256 value

arr 2

```
0: uint256: 30
```



Deployed Contracts

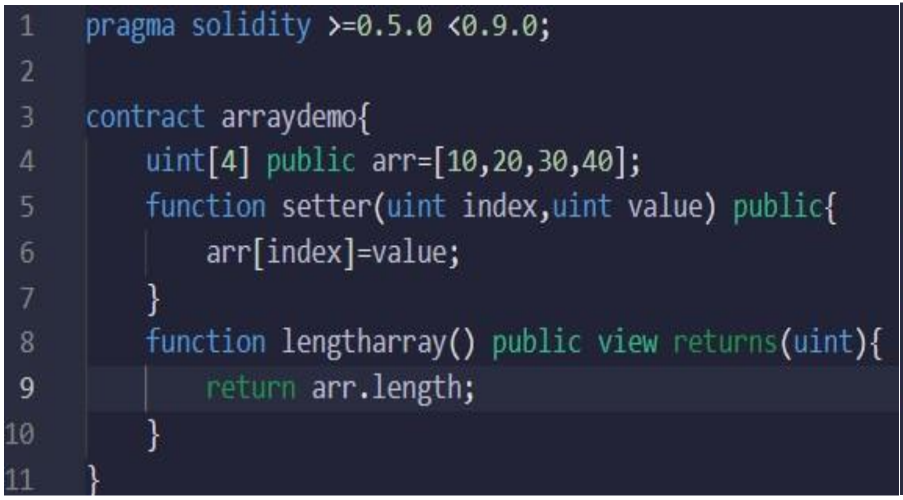
ARRAYDEMO AT 0XFFA...75D16 (

Balance: 0 ETH

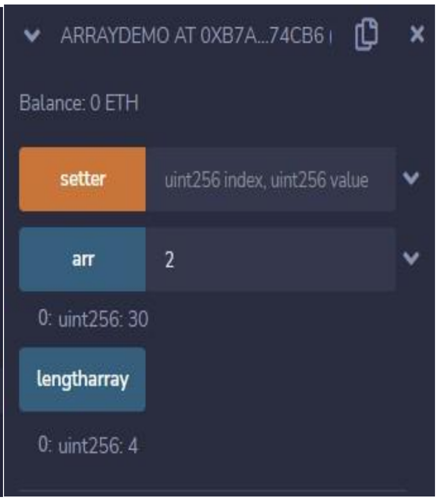
setter 2,999

arr 2

```
0: uint256: 999
```



```
pragma solidity >=0.5.0 <0.9.0;

contract arraydemo{
    uint[4] public arr=[10,20,30,40];
    function setter(uint index,uint value) public{
        arr[index]=value;
    }
    function lengtharray() public view returns(uint){
        return arr.length;
    }
}
```



ARRAYDEMO AT 0XB7A...74CB6 (

Balance: 0 ETH

setter uint256 index, uint256 value

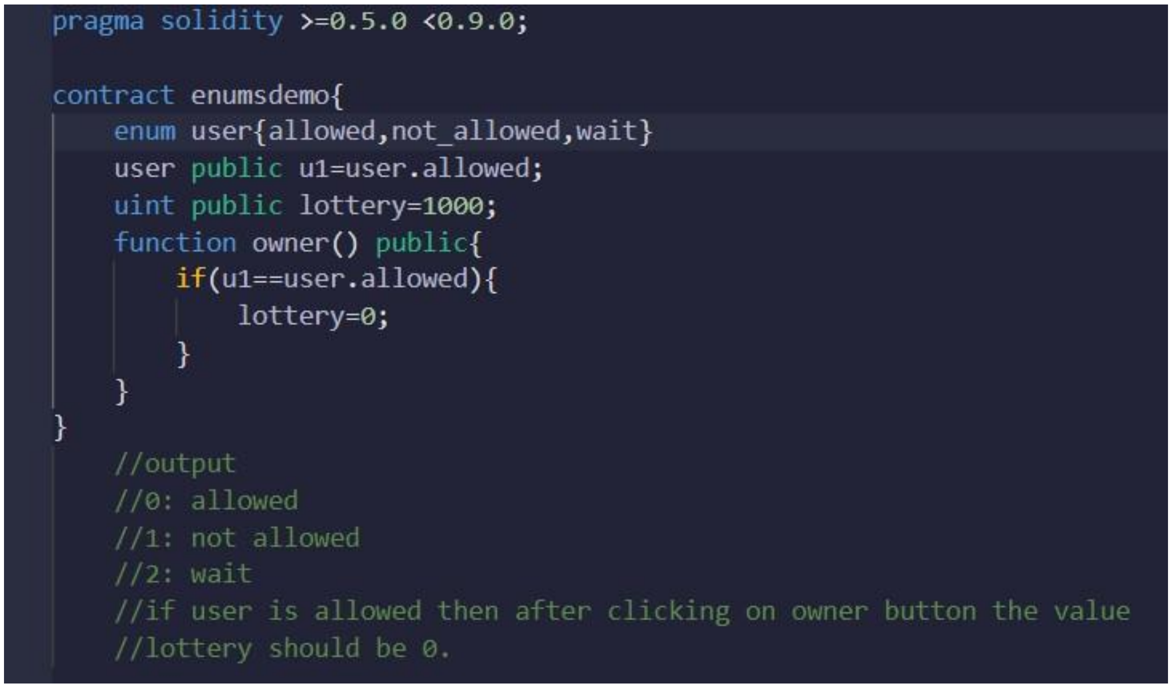
arr 2

```
0: uint256: 30
```

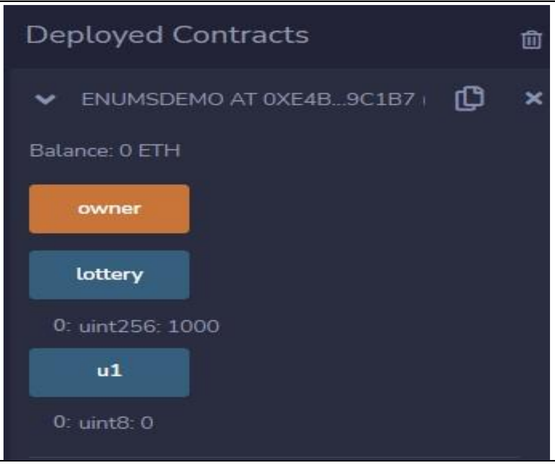
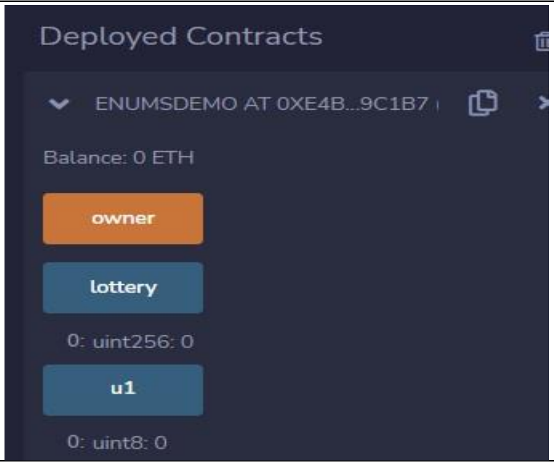
lengtharray

```
0: uint256: 4
```

3. ENUMS:

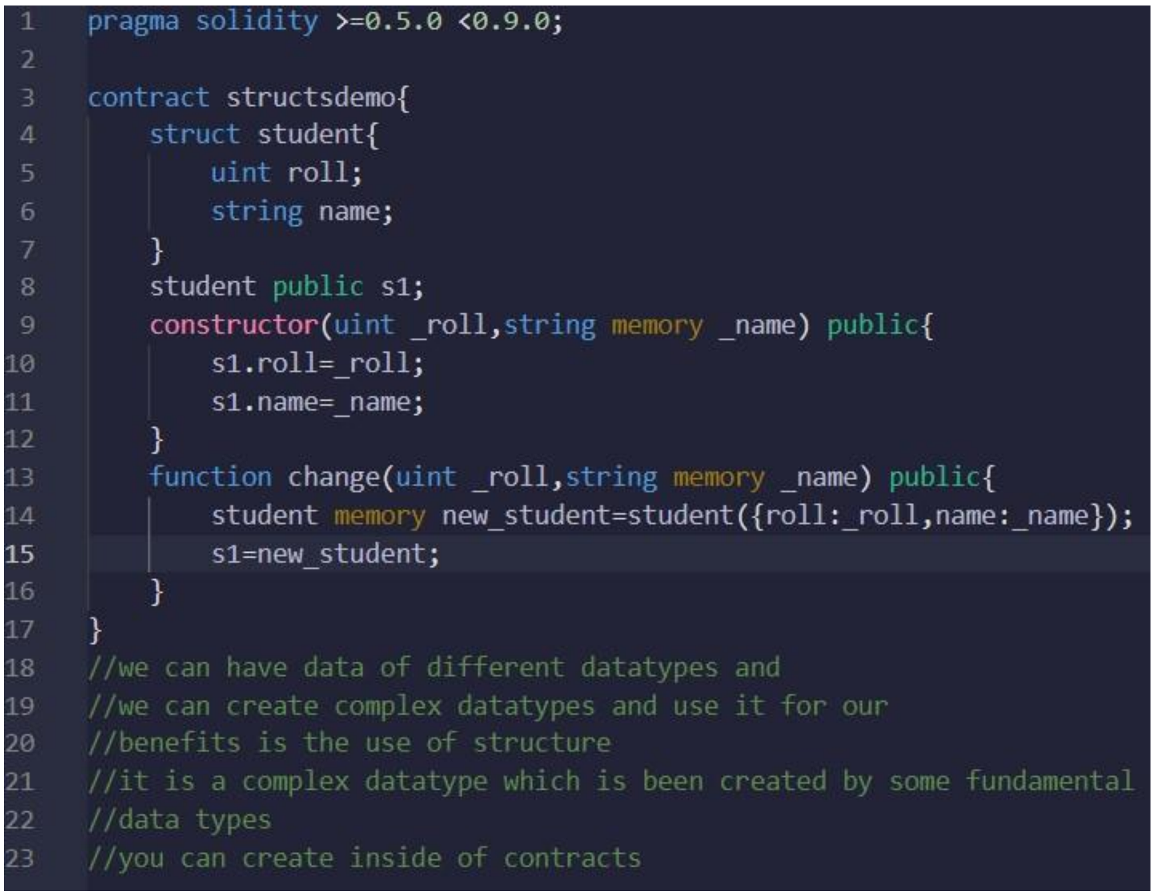


```
pragma solidity >=0.5.0 <0.9.0;

contract enumsdemo{
    enum user{allowed,not_allowed,wait}
    user public u1=user.allowed;
    uint public lottery=1000;
    function owner() public{
        if(u1==user.allowed){
            lottery=0;
        }
    }
}
    //output
    //0: allowed
    //1: not allowed
    //2: wait
    //if user is allowed then after clicking on owner button the value
    //lottery should be 0.
```

Before the lottery claim	After lottery claim
 Deployed Contracts ENUMSDEMO AT 0XE4B...9C1B7 Balance: 0 ETH owner lottery 0: uint256: 1000 u1 0: uint8: 0	 Deployed Contracts ENUMSDEMO AT 0XE4B...9C1B7 Balance: 0 ETH owner lottery 0: uint256: 0 u1 0: uint8: 0

Enums should be used for short amount of data

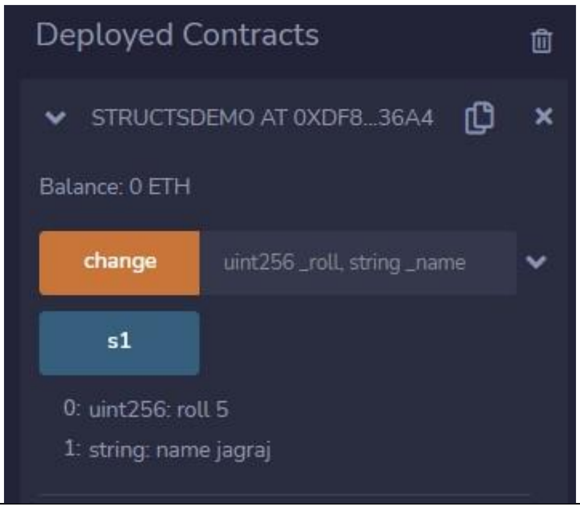
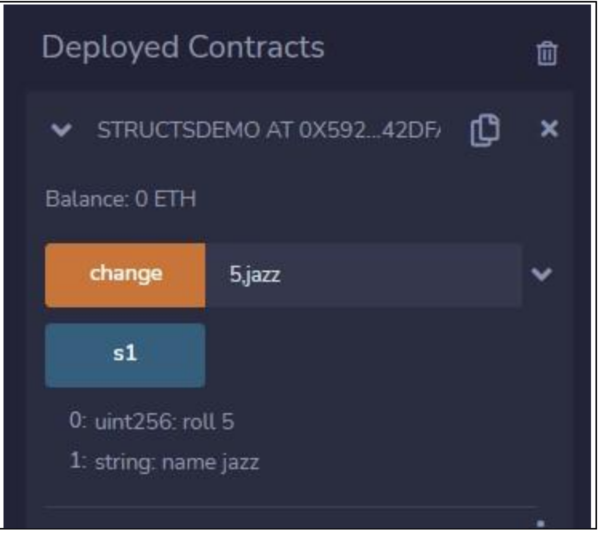
4. Structs:



```
pragma solidity >=0.5.0 <0.9.0;

contract structsdemo{
    struct student{
        uint roll;
        string name;
    }
    student public s1;
    constructor(uint _roll,string memory _name) public{
        s1.roll=_roll;
        s1.name=_name;
    }
    function change(uint _roll,string memory _name) public{
        student memory new_student=student({roll:_roll,name:_name});
        s1=new_student;
    }
}
//we can have data of different datatypes and
//we can create complex datatypes and use it for our
//benefits is the use of structure
//it is a complex datatype which is been created by some fundamental
//data types
//you can create inside of contracts
```

OUTPUT:

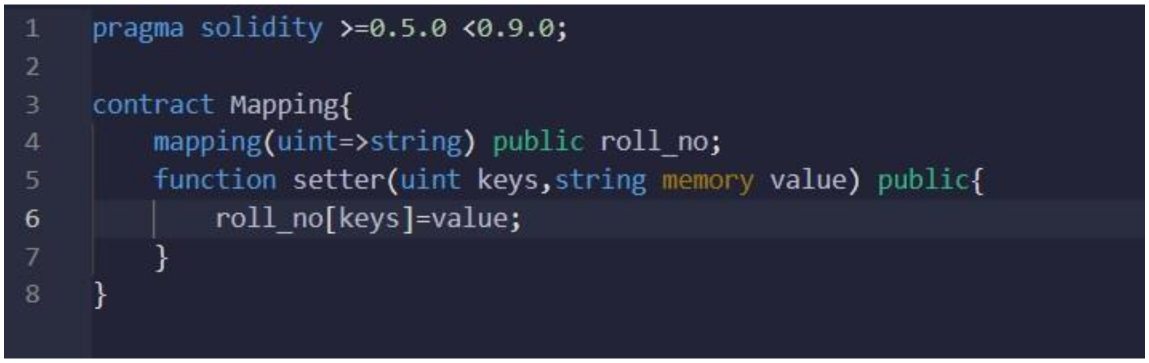
BEFORE UPDATE	AFTER UPDATE

Deployed Contracts
STRUCTSDEMO AT 0XDF8...36A4
Balance: 0 ETH
change uint256 _roll, string _name
s1
```
0: uint256: roll 5
1: string: name jagraj
```

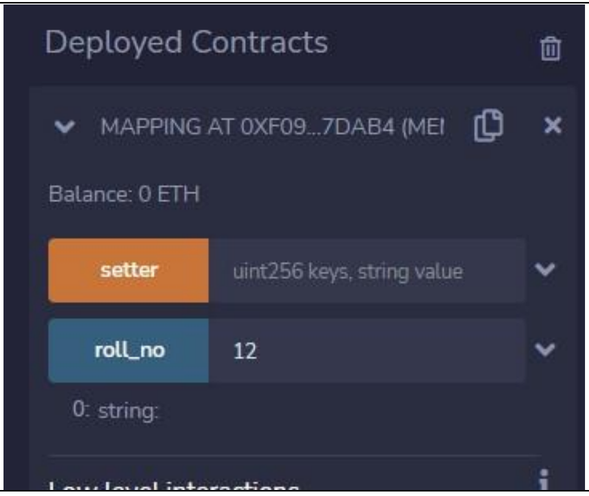
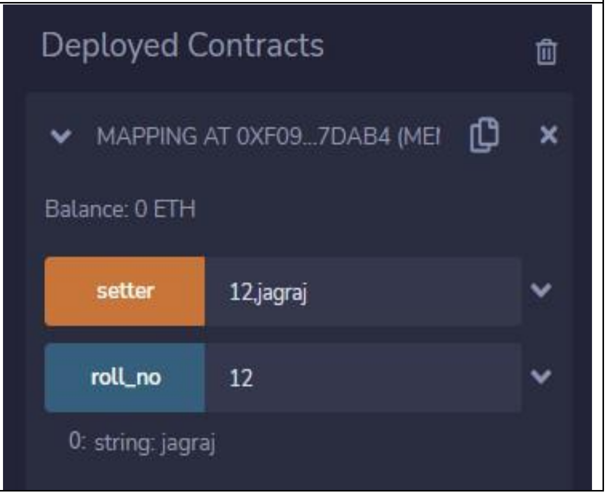
Deployed Contracts
STRUCTSDEMO AT 0X592...42DF
Balance: 0 ETH
change 5,jazz
s1
```
0: uint256: roll 5
1: string: name jazz
```

5. MAPPING:



```
pragma solidity >=0.5.0 <0.9.0;

contract Mapping{
    mapping(uint=>string) public roll_no;
    function setter(uint keys,string memory value) public{
        roll_no[keys]=value;
    }
}
```

OUTPUT:

BEFORE SETTING UP ANY VALUE	AFTER SETTING AN VALUE
	

Deployed Contracts
MAPPING AT 0XF09...7DAB4 (MEI
Balance: 0 ETH
setter uint256 keys, string value
roll_no 12
```
0: string:
```
Low level interactions

Deployed Contracts
MAPPING AT 0XF09...7DAB4 (MEI
Balance: 0 ETH
setter 12,jagraj
roll_no 12
```
0: string: jagraj
```

Why do we use mapping: Continuous storage is not required if data is stored at 100th loc then all the places before that will be wasted. But with mapping this problem is resolved.

PRACTICAL NO: 06

AIM: Implement and demonstrate the use of following in solidity:
1. Functions
2. View Functions
3. Pure Functions 4. Payable Function.

THEORY:

WHAT IS FUNCTION:

A function is basically a group of code that can be reused anywhere in the program, which generally saves the excessive use of memory and decreases the runtime of the program. Creating a function reduces the need of writing the same code over and over again. With the help of functions, a program can be divided into many small pieces of code for better understanding and managing.
In Solidity a function is generally defined by using the function keyword, followed by the name of the function which is unique and does not match with any of the reserved keywords. A function can also have a list of parameters containing the name and data type of the parameter. The return value of a function is optional but in solidity, the return type of the function is defined at the time of declaration.

SYNTAX:
```
function  function_name(parameter_list)  scope  returns(return_type)  {
// block of code
}
```

VIEW FUNCTION:

The view functions are read-only function, which ensures that state variables cannot be modified after calling them. If the statements which modify state variables, emitting events, creating other contracts, using selfdestruct method, transferring ethers via calls, calling a function which is not 'view or pure', using low-level calls, etc are present in view functions then the compiler throw a warning in such cases. By default, a get method is view function.

PURE FUNCTION:

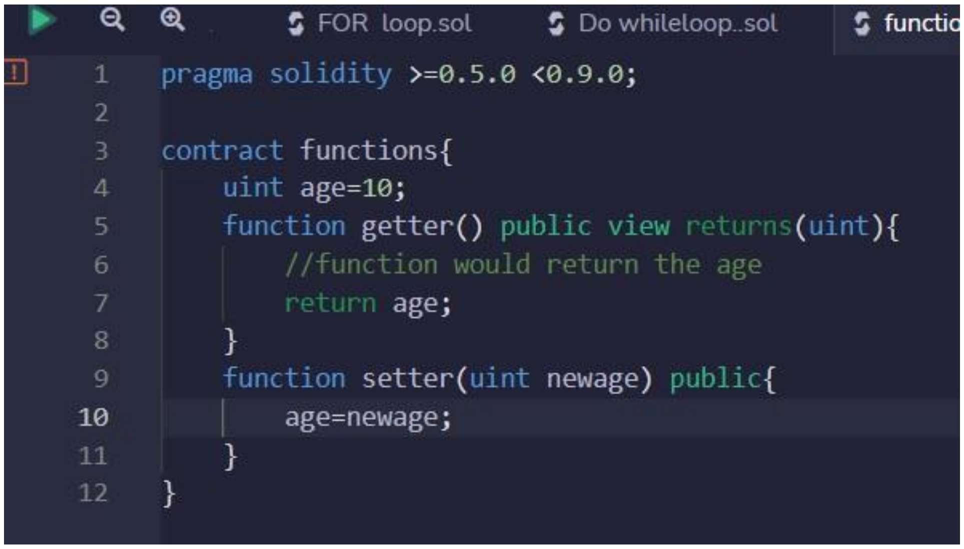
The pure functions do not read or modify the state variables, which returns the values only using the parameters passed to the function or local variables present in it. If the statements which read the state variables, access the address or balance, accessing any global variable block or msg, calling a function which is not pure, etc are present in pure functions then the compiler throws a warning in such cases.

PAYABLE FUNCTION:

payable function is a function that can receive ether. It provides the developer with the opportunity to respond to an ether deposit for record-keeping or any additional necessary logic.

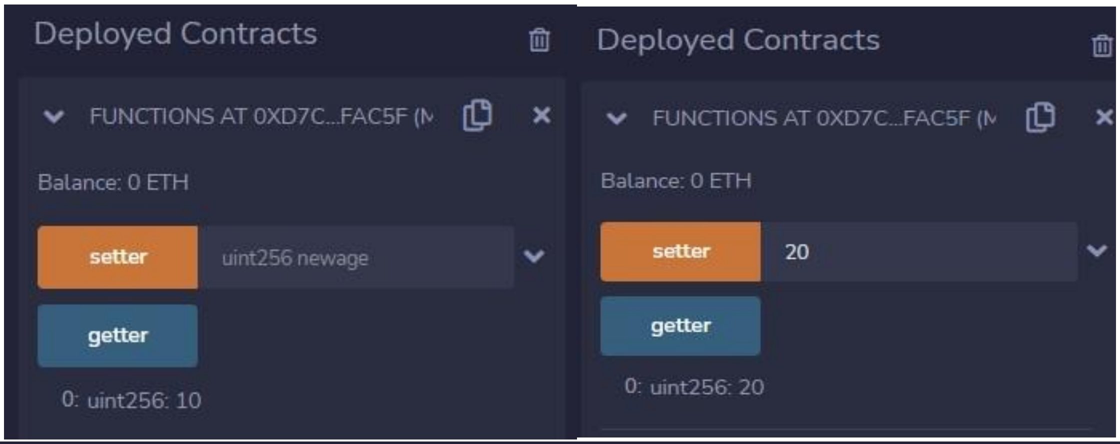
CODE:

FUNCTION1:



```
pragma solidity >=0.5.0 <0.9.0;

contract functions{
    uint age=10;
    function getter() public view returns(uint){
        //function would return the age
        return age;
    }
    function setter(uint newage) public{
        age=newage;
    }
}
```

OUTPUT:

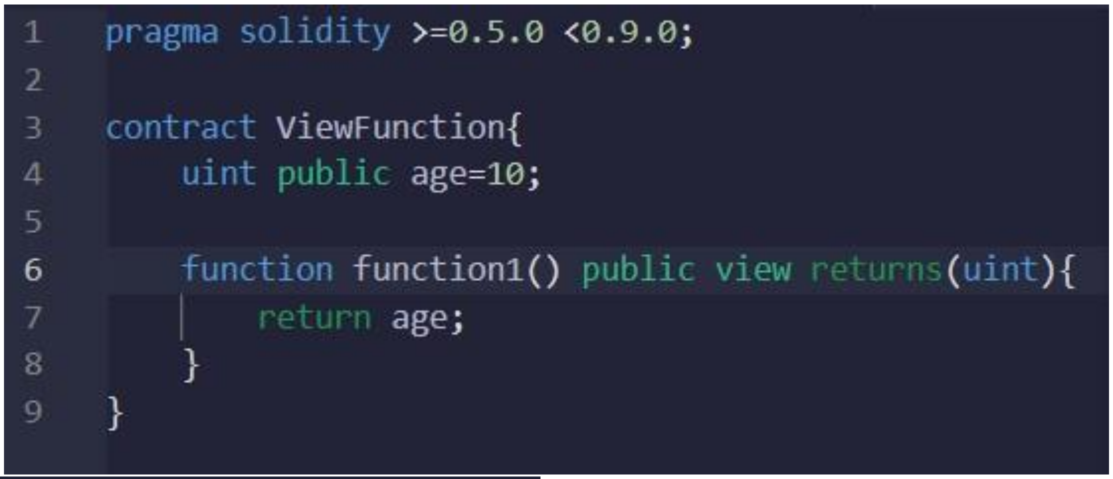


```
Deployed Contracts
FUNCTIONS AT 0XD7C...FAC5F (M
Balance: 0 ETH
setter  uint256 newage
getter
0: uint256: 10
```

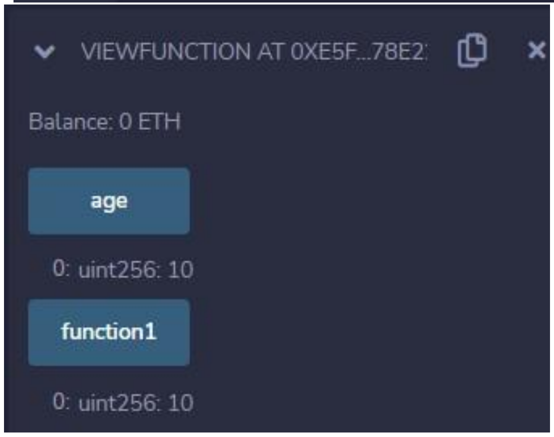
```
Deployed Contracts
FUNCTIONS AT 0XD7C...FAC5F (M
Balance: 0 ETH
setter  20
getter
0: uint256: 20
```

//instead of writing a getter function we can write public keyword onto the variable age it will be functioning same

VIEW FUNCTIONS:

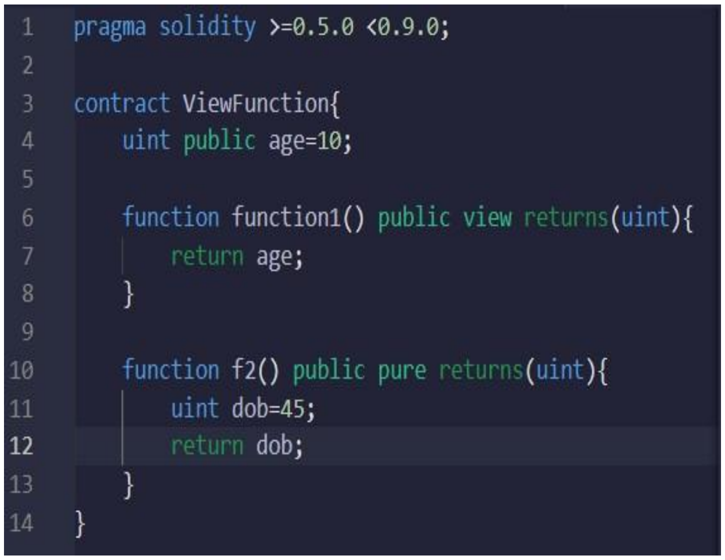


```
pragma solidity >=0.5.0 <0.9.0;

contract ViewFunction{
    uint public age=10;

    function function1() public view returns(uint){
        return age;
    }
}
```

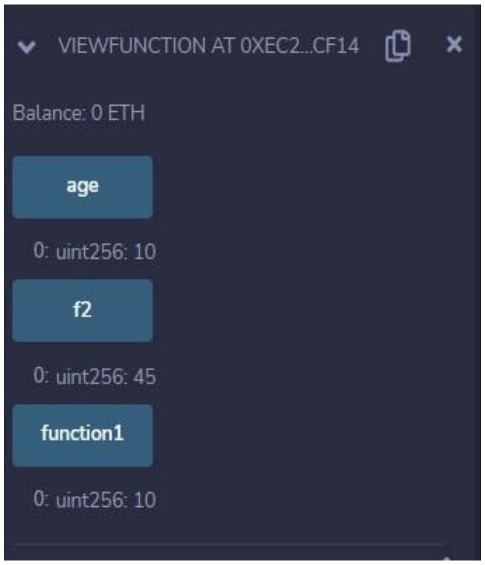


```
VIEWFUNCTION AT 0XE5F...78E2:
Balance: 0 ETH
age
0: uint256: 10
function1
0: uint256: 10
```

PURE FUNCTION



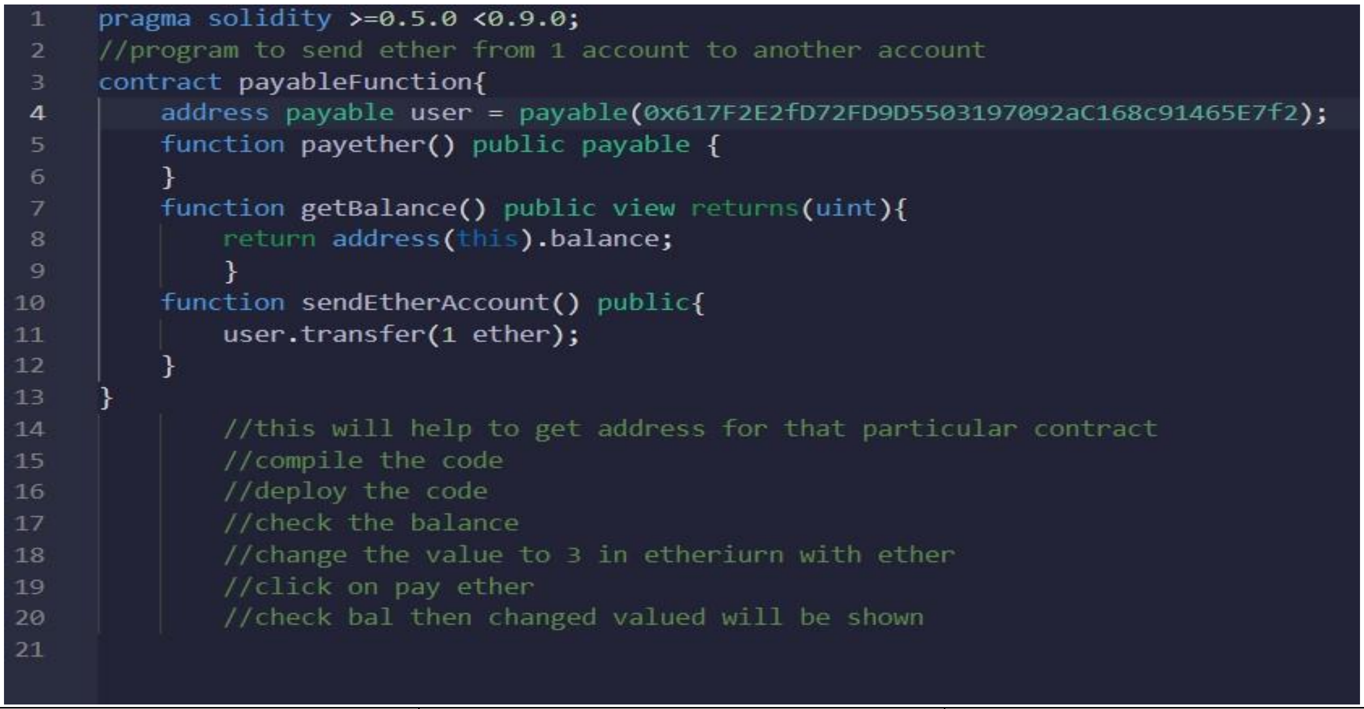
```
pragma solidity >=0.5.0 <0.9.0;

contract ViewFunction{
    uint public age=10;

    function function1() public view returns(uint){
        return age;
    }

    function f2() public pure returns(uint){
        uint dob=45;
        return dob;
    }
}
```



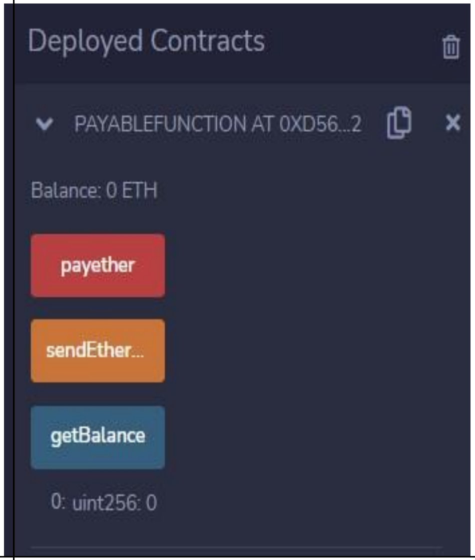
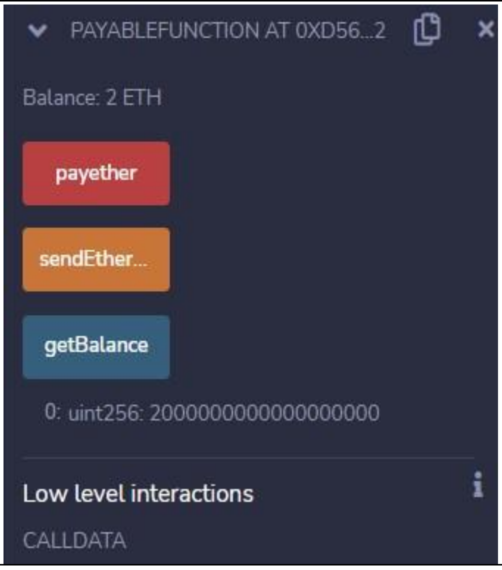
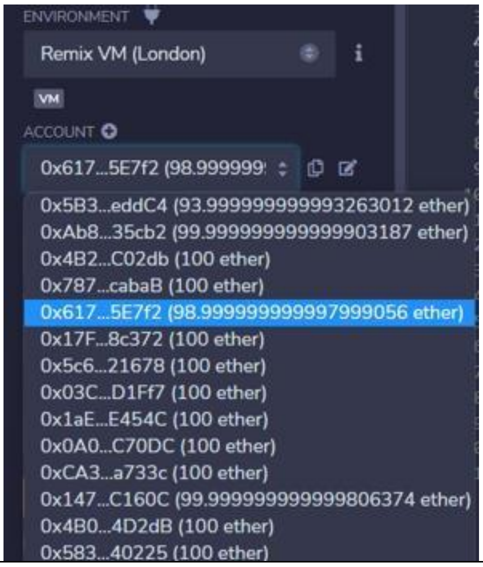
```
VIEWFUNCTION AT 0XEC2...CF14
Balance: 0 ETH
age
0: uint256: 10
f2
0: uint256: 45
function1
0: uint256: 10
```

PAYABLE FUNCTIONS:

Sahil Kilje
Roll No. 242



```
pragma solidity >=0.5.0 <0.9.0;
//program to send ether from 1 account to another account
contract payableFunction{
    address payable user = payable(0x617F2E2fD72FD9D5503197092aC168c91465E7f2);
    function payether() public payable {
    }
    function getBalance() public view returns(uint){
        return address(this).balance;
        }
    function sendEtherAccount() public{
        user.transfer(1 ether);
    }
}
        //this will help to get address for that particular contract
        //compile the code
        //deploy the code
        //check the balance
        //change the value to 3 in etheriurn with ether
        //click on pay ether
        //check bal then changed valued will be shown

```

1st op get balance	2nd op when ether value is selected and pay ether button is pressed and after that getbalance button is pressed.	3rd op when the user whose address is specified send the ether. Here we can see from the address of user we specified the ether got deducted and transferred to us.
 Deployed Contracts PAYABLEFUNCTION AT 0XD56...2 Balance: 0 ETH payether sendEther... getBalance 0: uint256: 0	 PAYABLEFUNCTION AT 0XD56...2 Balance: 2 ETH payether sendEther... getBalance 0: uint256: 2000000000000000000 Low level interactions CALLDATA	 ENVIRONMENT Remix VM (London) VM ACCOUNT 0x617...5E7f2 (98.999999 0x5B3...eddC4 (93.99999999993263012 ether) 0xAb8...35cb2 (99.999999999999903187 ether) 0x4B2...C02db (100 ether) 0x787...cabaB (100 ether) 0x617...5E7f2 (98.999999999997999056 ether) 0x17F...8c372 (100 ether) 0x5c6...21678 (100 ether) 0x03C...D1Ff7 (100 ether) 0x1aE...E454C (100 ether) 0x0A0...C70DC (100 ether) 0xCA3...a733c (100 ether) 0x147...C160C (99.999999999999806374 ether) 0x4B0...4D2dB (100 ether) 0x583...40225 (100 ether)

PRACTICAL NO:07

AIM: Implement and demonstrate the use of following in solidity:
1. Function Overloading
2. Mathematical Functions
3. Cryptographic Functions

THEORY:

FUNCTION OVERLOADING:

Object-oriented programming has a function overloading feature, which allows two or more functions to have the same name but distinct parameters.
In Solidity, we can have many definitions for similar function names within the same scope. The kinds and number of arguments in the argument list must differ in the definition of the function. Function declarations that differ only in return type cannot be overloaded.

MATHEMATICAL FUNCTIONS:

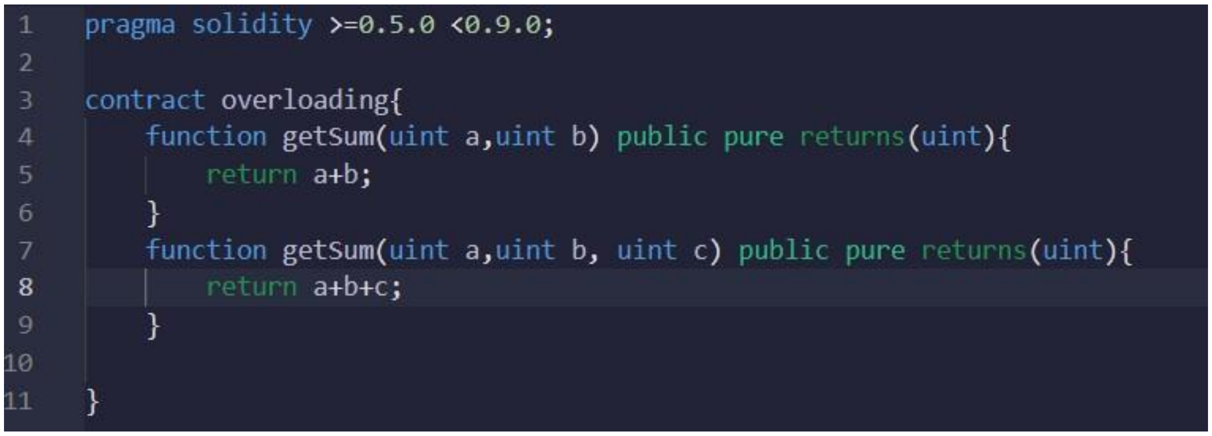
Solidity provides inbuilt mathematical functions.
Following are heavily used methods −

• addmod(uint x, uint y, uint k) returns (uint) − computes (x + y) % k where the addition is performed with arbitrary precision and does not wrap around at 2^{256}. • mulmod(uint x, uint y, uint k) returns (uint) − computes (x * y) % k where the addition is performed with arbitrary precision and does not wrap around at 2^{256}.

CRYPTOGRAPHIC FUNCTIONS:
A cryptographic hash function is an algorithm that takes an arbitrary amount of data as input and produces the enciphered text of fixed size. Even a slight change in the input gives a completely different output.
Solidity provides inbuilt cryptographic functions as well. Following are important methods − keccak256(bytes memory) returns (bytes32) − computes the Keccak-256 hash of the input.

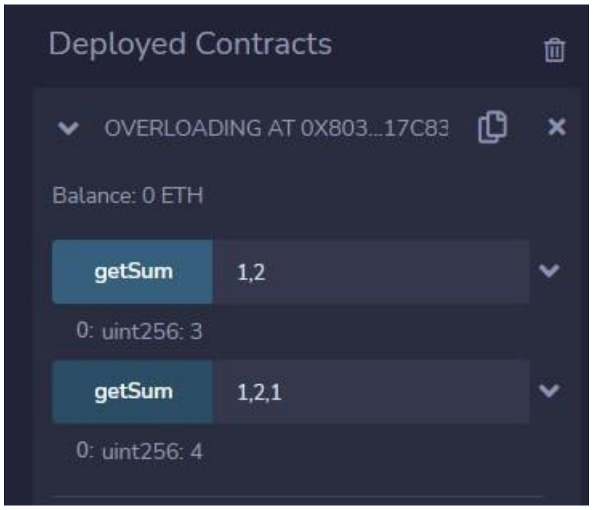
CODE:

FUNCTION OVERLOADING:



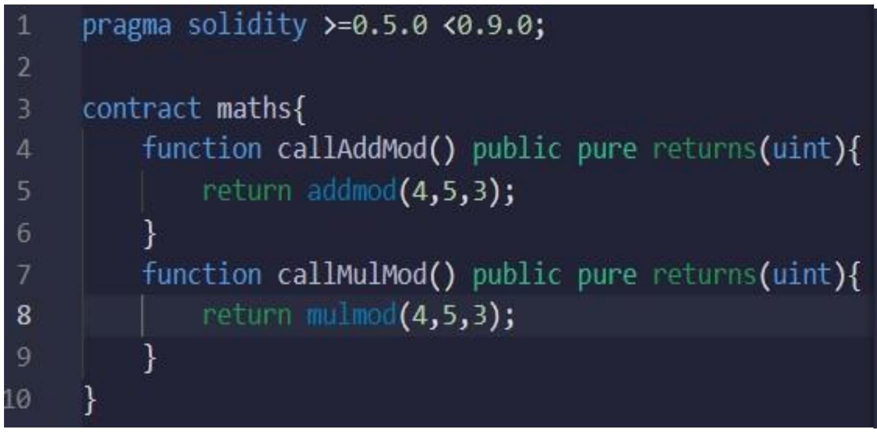
```
pragma solidity >=0.5.0 <0.9.0;

contract overloading{
    function getSum(uint a,uint b) public pure returns(uint){
        return a+b;
    }
    function getSum(uint a,uint b, uint c) public pure returns(uint){
        return a+b+c;
    }

}
```

OUTPUT:

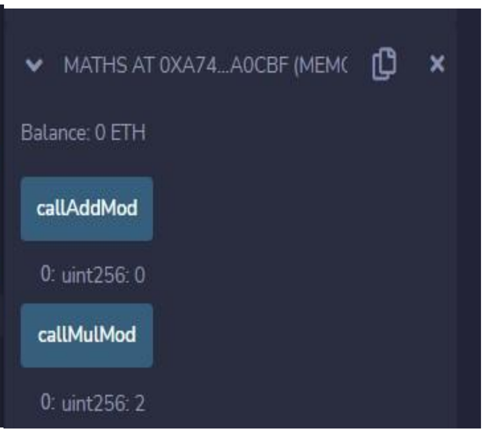


```
Deployed Contracts
OVERLOADING AT 0X803...17C83
Balance: 0 ETH
getSum  1,2
0: uint256: 3
getSum  1,2,1
0: uint256: 4
```

MATHEMATICAL FUNCTION:

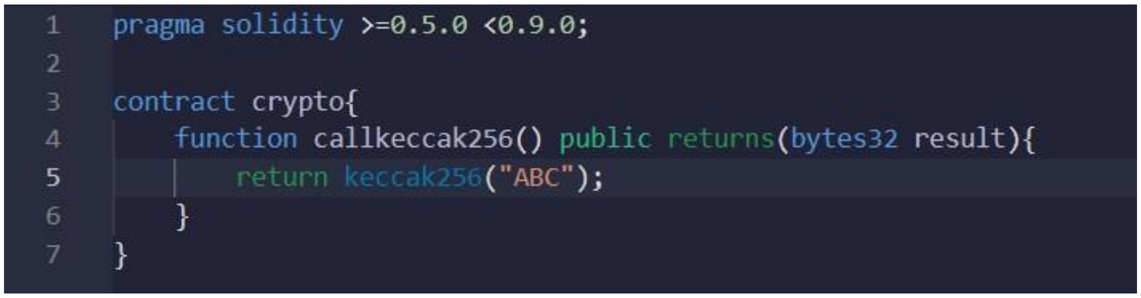


```
pragma solidity >=0.5.0 <0.9.0;

contract maths{
    function callAddMod() public pure returns(uint){
        return addmod(4,5,3);
    }
    function callMulMod() public pure returns(uint){
        return mulmod(4,5,3);
    }
}
```



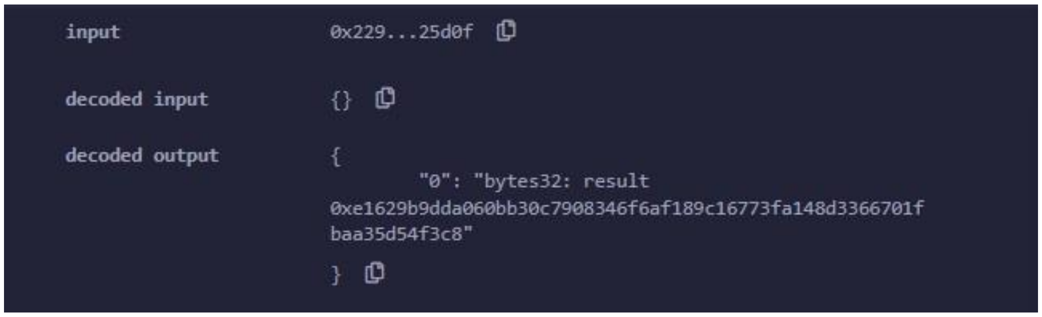
```
MATHS AT 0XA74...A0CBF (MEM(
Balance: 0 ETH
callAddMod
0: uint256: 0
callMulMod
0: uint256: 2
```

CRYPTO FUNCTION:

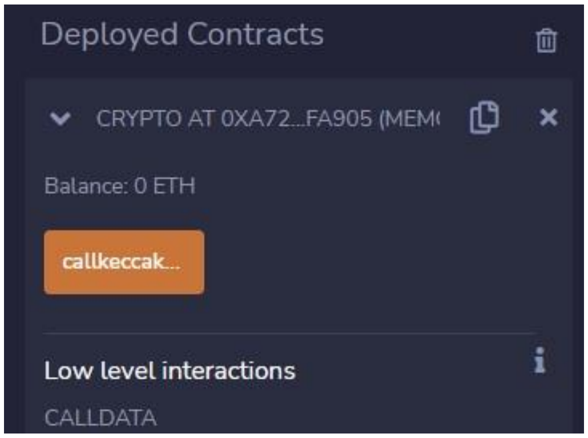


```
pragma solidity >=0.5.0 <0.9.0;

contract crypto{
    function callkeccak256() public returns(bytes32 result){
        return keccak256("ABC");
    }
}
```

OUTPUT:



```
input               0x229...25d0f

decoded input       {}

decoded output      {
                            "0": "bytes32: result
                    0xe1629b9dda060bb30c7908346f6af189c16773fa148d3366701f
                    baa35d54f3c8"
                    }
```



Deployed Contracts

CRYPTO AT 0XA72...FA905 (MEM(

Balance: 0 ETH

Low level interactions

CALLDATA

PRACTICAL NO: 08

AIM: Implement and demonstrate the use of following in solidity:

1. Pubic
2. Private
3. Internal
4. External
5. Contract
6. Constructors

THEORY:

VISIBILITY:

In Solidity, you can control who has access to the functions and state variables in your contract and how they interact with them. This concept is known as visibility.
Following are various visibility quantifiers for functions/state variables of a contract.

- external − External functions are meant to be called by other contracts. They cannot be used for internal call. To call external function within contract this.function_name() call is required. State variables cannot be marked as external.
- public − Public functions/ Variables can be used both externally and internally. For public state variable, Solidity automatically creates a getter function.
- internal − Internal functions/ Variables can only be used internally or by derived contracts.
- private − Private functions/ Variables can only be used internally and not even by derived contracts.

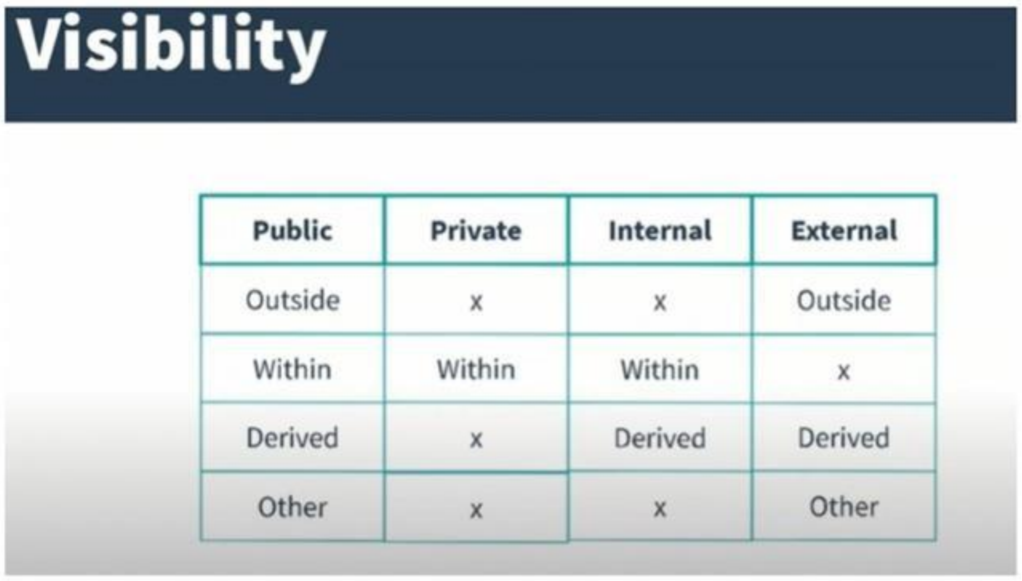
CONTRACT:
Solidity's code is encapsulated in contracts which means a contract in Solidity is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain. A contract is a fundamental block of building an application on Ethereum. Syntax

```
contract Storage{  //Functions
and Data
}
```

CONSTRUCTOR

Constructor is a special function declared using **constructor** keyword. It is an optional funtion and is used to initialize state variables of a contract. Following are the key characteristics of a constructor.

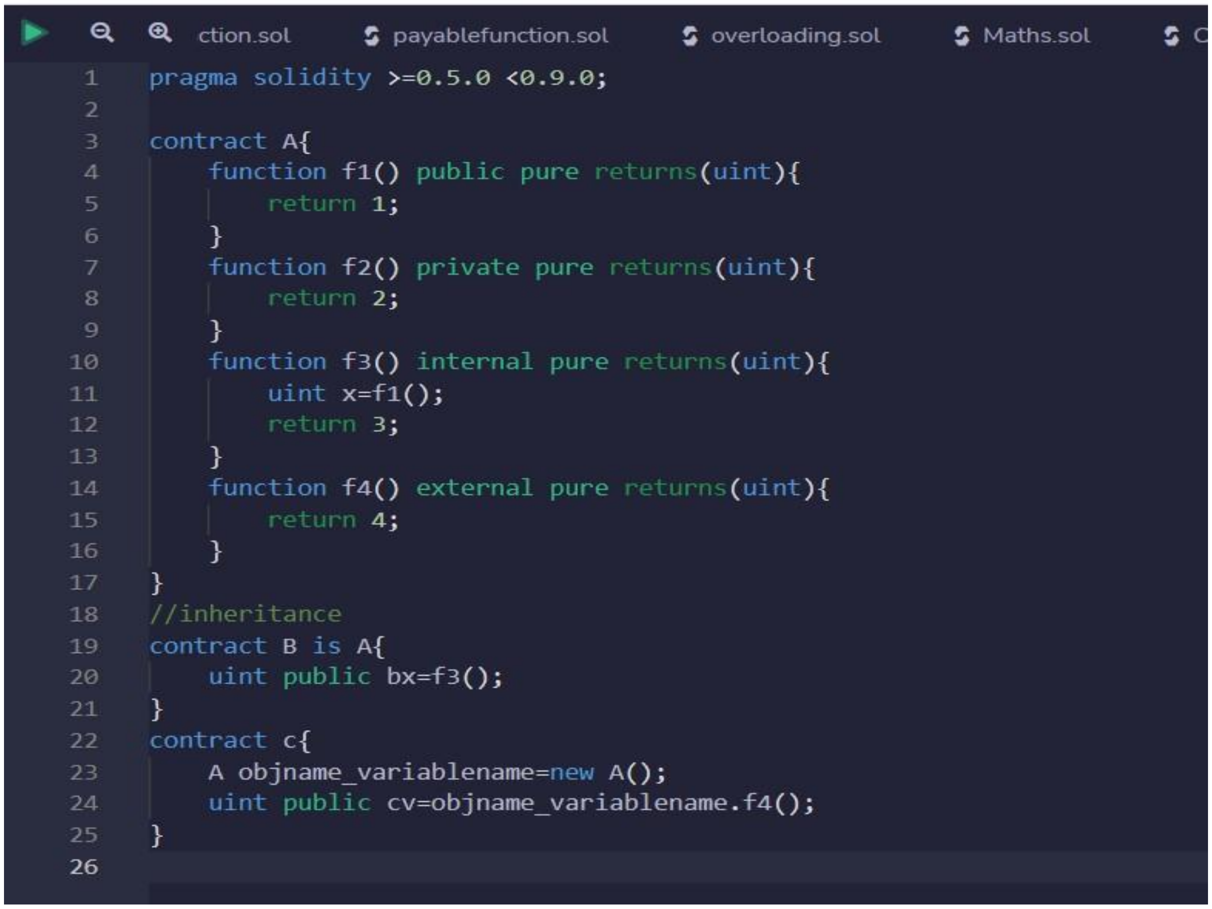
- A contract can have only one constructor.
- A constructor code is executed once when a contract is created and it is used to initialize contract state.
- After a constructor code executed, the final code is deployed to blockchain. This code includes public functions and code reachable through public functions. Constructor code or any internal method used only by constructor are not included in final code.
- A constructor can be either public or internal.
- An internal constructor marks the contract as abstract.
- In case, no constructor is defined, a default constructor is present in the contract.



Visibility

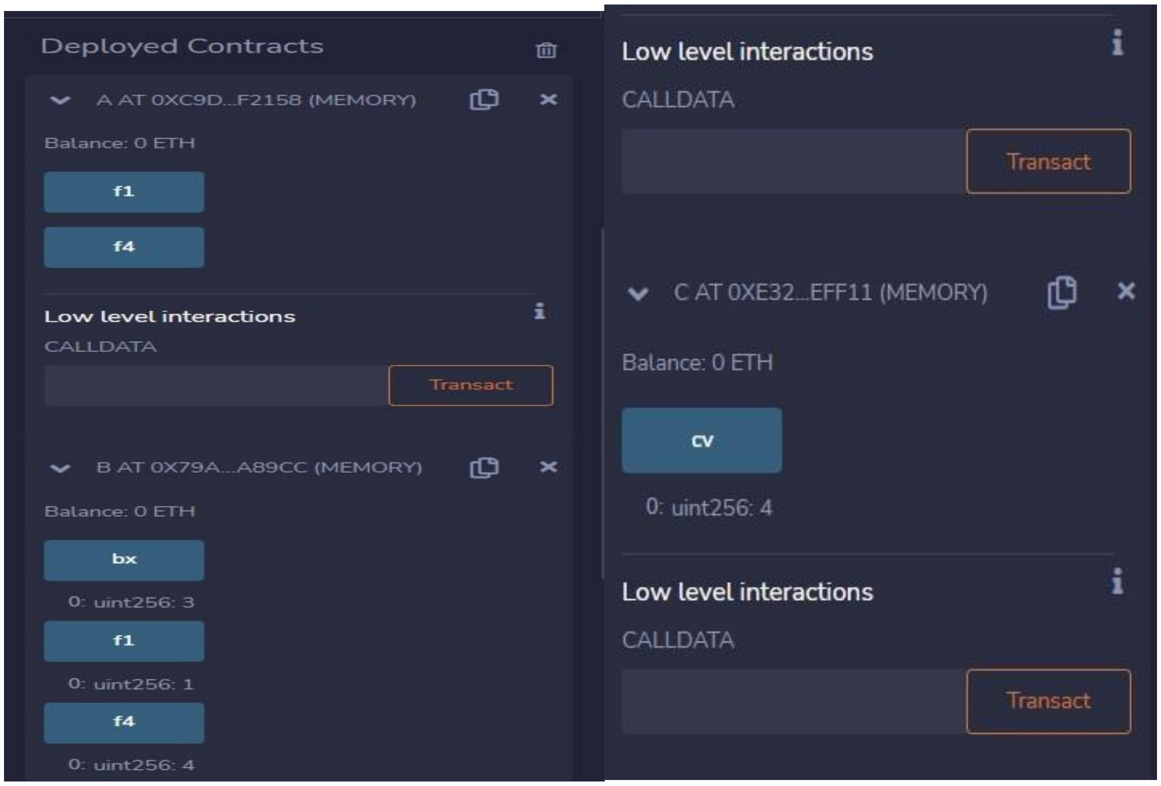
Public	Private	Internal	External
Outside	x	x	Outside
Within	Within	Within	x
Derived	x	Derived	Derived
Other	x	x	Other

CODE:

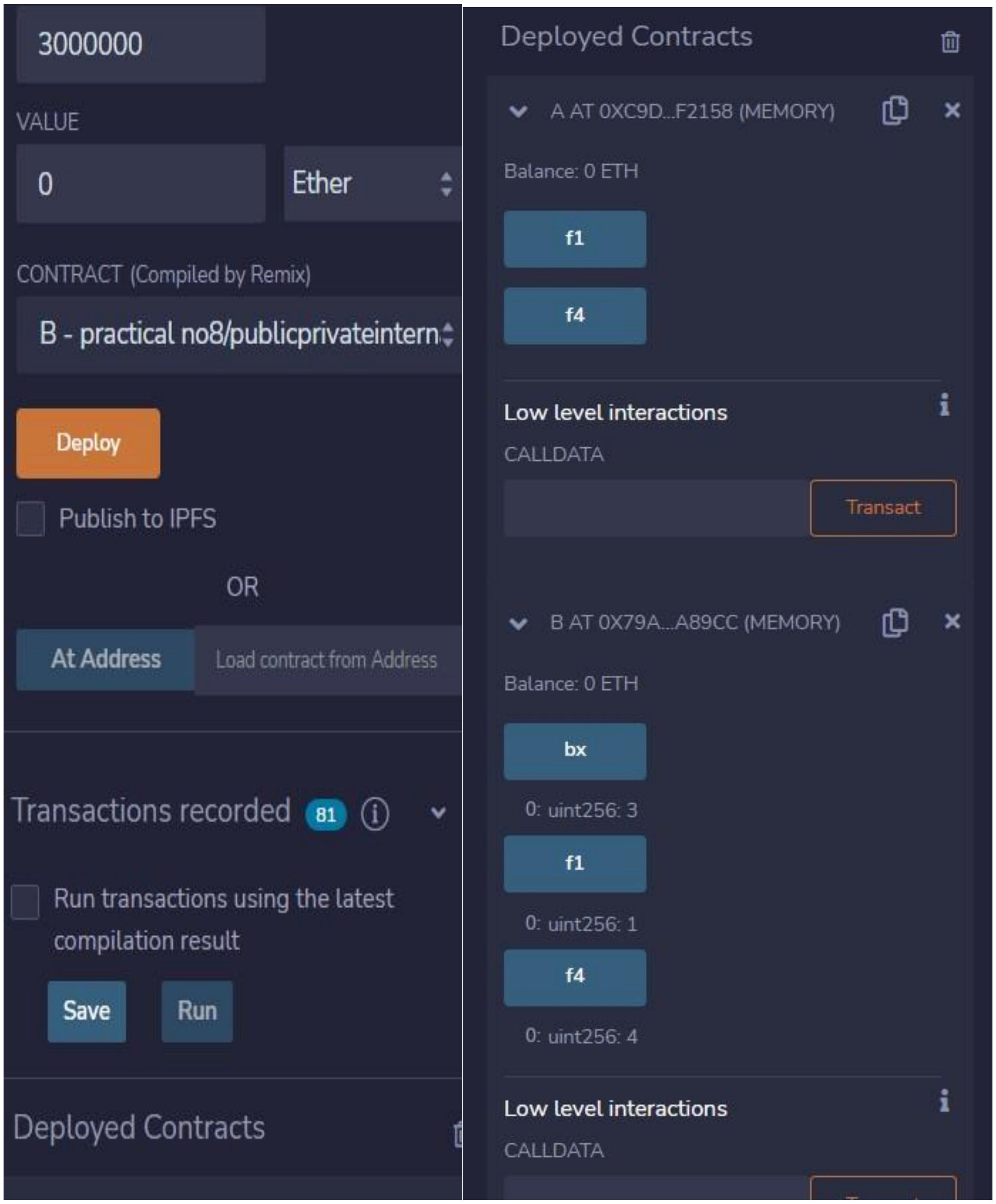


```
pragma solidity >=0.5.0 <0.9.0;

contract A{
    function f1() public pure returns(uint){
        return 1;
    }
    function f2() private pure returns(uint){
        return 2;
    }
    function f3() internal pure returns(uint){
        uint x=f1();
        return 3;
    }
    function f4() external pure returns(uint){
        return 4;
    }
}
//inheritance
contract B is A{
    uint public bx=f3();
}
contract c{
    A objname_variablename=new A();
    uint public cv=objname_variablename.f4();
}
```

OUTPUT:



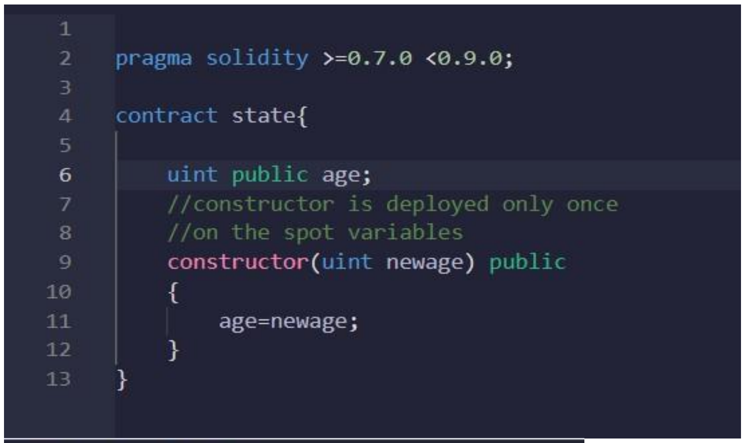
```
bx
0: uint256: 3
f1
0: uint256: 1
f4
0: uint256: 4
cv
0: uint256: 4
```



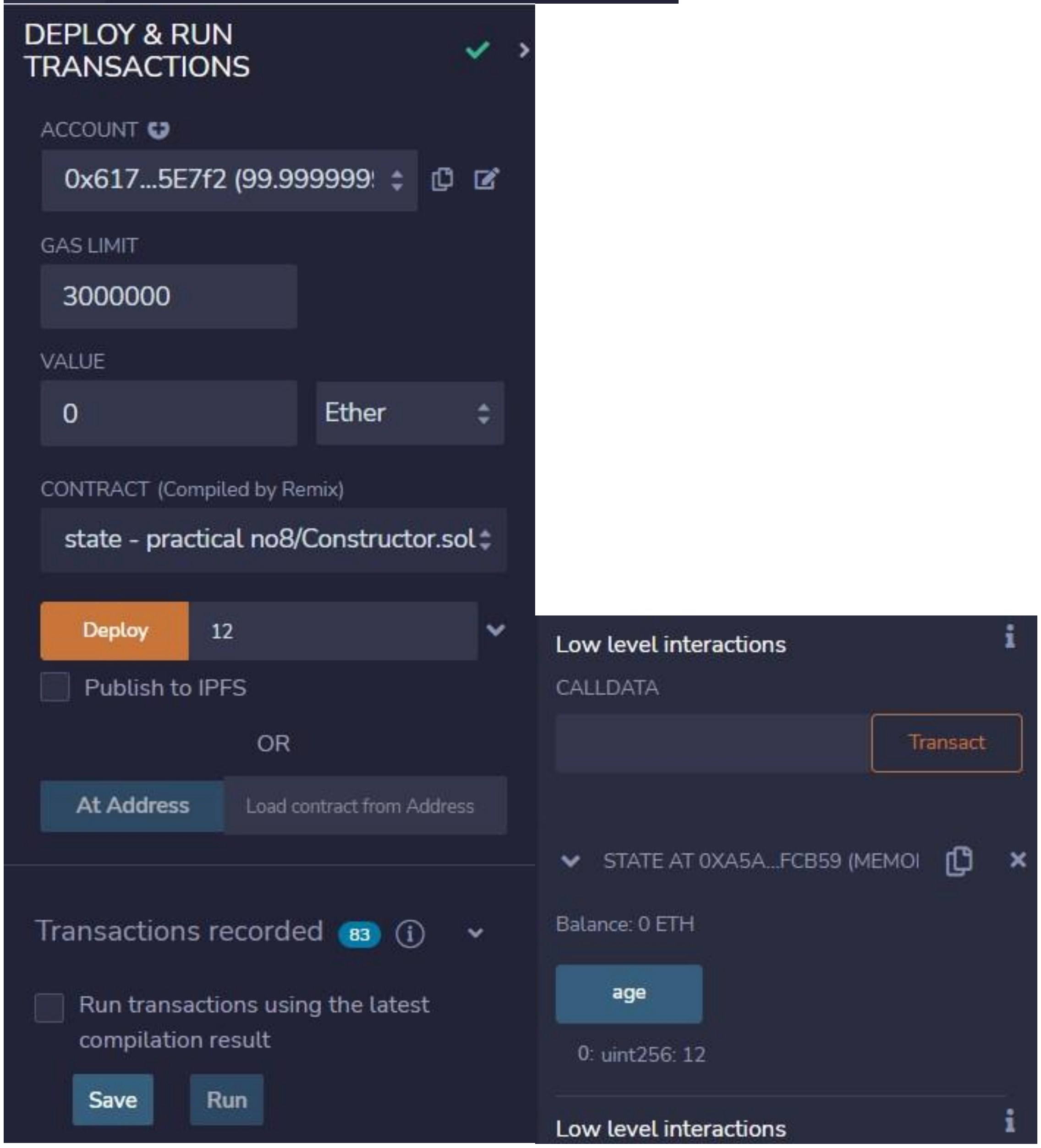
```
3000000
VALUE
0    Ether
CONTRACT (Compiled by Remix)
B - practical no8/publicprivateintern
Deploy
Publish to IPFS
OR
At Address    Load contract from Address
Transactions recorded 81
Run transactions using the latest compilation result
Save    Run
Deployed Contracts
```

```
Deployed Contracts
A AT 0XC9D...F2158 (MEMORY)
Balance: 0 ETH
f1
f4
Low level interactions
CALLDATA
Transact

B AT 0X79A...A89CC (MEMORY)
Balance: 0 ETH
bx
0: uint256: 3
f1
0: uint256: 1
f4
0: uint256: 4
Low level interactions
CALLDATA
```

CONSTRUCTOR:



```

pragma solidity >=0.7.0 <0.9.0;

contract state{

    uint public age;
    //constructor is deployed only once
    //on the spot variables
    constructor(uint newage) public
    {
        age=newage;
    }
}
```



DEPLOY & RUN TRANSACTIONS

ACCOUNT
0x617...5E7f2 (99.999999

GAS LIMIT
3000000

VALUE
0 Ether

CONTRACT (Compiled by Remix)
state - practical no8/Constructor.sol

Deploy 12

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 83

Run transactions using the latest compilation result

Save Run

Low level interactions
CALLDATA
Transact

STATE AT 0XA5A...FCB59 (MEMOI

Balance: 0 ETH

age
0: uint256: 12

Low level interactions