1. Your first program in Solidity

Objective - In this challenge, we review some basic concepts that will get you started with this series. You will need to use the same (or similar) syntax to read input and write output in challenges throughout Codedamn.

Task - To complete this challenge, you must print Hello Solidity. Use str as a variable to store value.

You can do this :-)

Ans:-

pragma solidity >=0.5.0 < 0.9.0;

contract HelloSolidity{

string public str = "Hello Solidity";

}

Create functions

Task-

To complete this challenge, you have to create a state variable and initialised it with 10 and then you have to create a function returnStateVariable( ) which returns the value of the created state variable .

In the same way you have to create a local variable and initialised it with 20 and then you have to create a function returnLocalVariable( ) which returns the value of the created local variable.

You can do this :-)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity ^ 0.8;

contract task2{

uint8 a=10;

function returnStateVariable( ) public view returns (uint8) {

return a;

}

function returnLocalVariable( ) public pure returns (uint8){

uint8 b=20;

return b;

}

}

Pass an argument to function

Task

To complete this challenge, do the following.

1)Create a state variable of uint type (need not to be public).

2)Create a set ( ) and with one argument. And set the value of the created state variable with the value passed as an argument to the set ( ).

3)Then create another function get ( ) which returns the value of the state variable.

You can do this :-)

Ans:

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 < 0.9.0;

contract FunctionArgument{

uint num =10;

function set(uint \_item) public {

num=\_item;

}

function get() public view returns(uint){

return num;

}

}

Solidity Basics

Task

Create a function evaluate(int a , int b).

Subtract the difference of a and b from the sum of a and b.

Return the result of the above task from the evaluate()

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract day4 {

function evaluate(int256 a, int256 b) public pure returns (int256) {

return ((a + b) - (a - b));

}

}

Find remainder

Task

Create a function find(int a). Where a>0.

Return the remainder when a is divided by 3.

Note - Function will be public.

You can do this :)

Ans—

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract day5 {

function find(int256 a) public pure returns (int256) {

require(a>0,"a should not be negative");

return a % 3;

}

}

Find average

Task

Create a function average(int a,int b,int c). Where a>0,b>0,c>0.

Return the average of a,b and c.

Note - Function will be public.

You can do this :)

Ans:

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract day6 {

function average(

int256 a,

int256 b,

int256 c

) public pure returns (int256) {

return (a + b + c) / 3;

}

}

Find the sum of digits

Task

Create a function digitSum(int n). Where n>0.

Return the sum of digit for n.

For Example - If n=345 then digitSum() must return 12 i.e. 3+4+5 = 12

If n=909 then digitSum() must return 18 i.e. 9+0+9 = 18

Note - Function will be public.

You can do this :)

Ans

// SPDX-License-Identifier: aIT

pragma solidity >=0.5.0 <0.9.0;

contract day1 {

function digitSum(int256 n) public pure returns (int256) {

int256 a;

int256 sum = 0;

while (n > 0) {

a = n % 10;

sum = sum + a;

n = n / 10;

}

return sum;

}

}

nth Term

Task

There is a series, S , where the next term is the sum of pervious three terms. Given the first three terms of the series, a ,b ,c and respectively, you have to output the nth term of the series.

S(n) = a for n=1

S(n) = b for n=2

S(n) = c for n=3

S(n) = S(n-1) + S(n-2) + S(n-3) for n>3

Create a function nthTerm(uint n, uint a, uint b, uint c) where n is the nth term to find and a,b,c are the three terms of the series.

For Example - If n=5 a=1 b=2 c=3 then nthTerm() must return 11 as S(1) = 1 S(2) = 2

S(3) = 3 S(4) = S(3) + S(2) + S(1) = 1+2+3 = 6 S(5) = S(4) + S(3) + S(2) = 6 + 3 + 2 = 11

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day2 {

function nthTerm(

uint256 n,

uint256 a,

uint256 b,

uint256 c

) public pure returns (uint256) {

uint256[100] memory arr;

uint256 i;

arr[1] = a;

arr[2] = b;

arr[3] = c;

for (i = 4; i <= n; i++) {

arr[i] = arr[i - 1] + arr[i - 2] + arr[i - 3];

}

return arr[n];

}

}

Prime Number

Task

A prime number is a whole number greater than 1. It has exactly two factors, that is, 1 and the number itself.

Create a function prime(uint n) . This prime() will check whether n is a prime number or not.

If n is a prime number then prime() returns 1 and n is not prime then prime() must return 0.

For Example - If n=103 then prime() must return 1 as 103 is a prime number. If n=200 then prime() must return 0 as 200 is not a prime number.

Note - Function will be public.

You can do this :)

Ans

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day3 {

function prime(uint256 n) public pure returns (uint256) {

if (n <= 1) {

return 0; //not prime

}

for (uint256 i = 2; i < n; i++) {

if (n % i == 0) {

return 0; //not prime

}

}

return 1; //prime

}

}

X raised to Y

Task

If y is a positive integer and x is any real number, then x^y corresponds to repeated multiplication x^y=x×x×⋯×x y times. We can call this “x raised to the power of y,” “x to the power of y,” or simply “x to the y.” Here, x is the base and y is the exponent or the power.

Create a function power(uint x,uint y) . This power() will calculate x raised to the power of y and return it.

For Example - If x=2 y=3 then power() must return 8 (2x2x2=8) If x=7 y=2 then power() must return 49 (7x7=49)

Note - Function will be public.

You can do this :)

Ans:

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day4 {

function power(uint256 base, uint256 exponent)

public

pure

returns (uint256)

{

uint256 value = 1;

while (exponent != 0) {

value \*= base;

--exponent;

}

return value;

}

}

Palindrome

Task

A palindrome number is a number that is same after reverse. For example 545, 151, 34543 etc.

Create a function palindrome(uint n) . Thispalindrome() will check whether n is a palindrome or not.

If n is a palindrome then palindrome() returns 1 and n is not palindrome then palindrome() must return 0.

For Example - If n=545 then palindrome() must return 1. If n=123 then palindrome() must return 0.

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day5 {

function palindrome(uint256 n) public pure returns (uint256) {

uint256 reversed = 0;

uint256 remainder;

uint256 original;

original = n;

while (n != 0) {

remainder = n % 10;

reversed = reversed \* 10 + remainder;

n /= 10;

}

if (original == reversed) {

return 1; //palindrome

} else {

return 0; //not palindrome

}

}

}

Reverse a Number

Task

Create a function reverseDigit(uint n) . This reverseDigit() will do the reversal of the number n.

For Example - If n=123 then reverseDigit() must return 321. If n=5342 then reverseDigit() must return 2435.

Note - Function will be public.

You can do this :)

Ans:

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day6 {

function reverseDigit(uint256 n) public pure returns (uint256) {

uint256 reverse = 0;

uint256 remainder;

while (n != 0) {

remainder = n % 10;

reverse = reverse \* 10 + remainder;

n /= 10;

}

return reverse;

}

}

Array Reversal

Task

Create a function reverseArray(array, length of array) . This reverseArray() will take two arguments - a dynamic uint type array and length of the array.

The reverseArray() will reverse the array. For Example - If array =[2,5,9,11,1] then reverseArray() will return [1,11,9,5,2] If array =[90,20,30,10] then reverseArray() will return [10,30,20,90]

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day1 {

function reverseArray(uint256[] memory arr, uint256 len)

public

pure

returns (uint256[] memory)

{

uint256 temp;

for (uint256 i = 0; i < len / 2; i++) {

temp = arr[i];

arr[i] = arr[len - i - 1];

arr[len - i - 1] = temp;

}

return arr;

}

}

Array to even

Task

Create a function even(array, length of array) . This even() will take two arguments - a dynamic uint type array and length of the array.

The even() will multiply each element of array with 2. For Example - If array =[2,5,11,1] then even() will return [4,10,22,2,] If array =[1,3,5,7] then even() will return [2,6,10,14]

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day2 {

function even(uint256[] memory arr, uint256 len)

public

pure

returns (uint256[] memory)

{

for (uint256 i = 0; i < len; i++) {

arr[i] = 2 \* arr[i];

}

return arr;

}

}

Array Sorting

Task

Create a function sort(array, length of array) . This sort() will take two arguments - a dynamic uint type array and length of the array.

The sort() will sort the array elements in ascending order. For Example - If array =[2,5,9,11,1] then sort() will return [1,2,5,9,11] If array =[90,20,30,10] then sort() will return [10,20,30,90]

Note - Function will be public.

You can do this :)

ANs:--

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day3 {

function sort(int256[] memory array, uint256 size)

public

pure

returns (int256[] memory)

{

for (uint256 step = 0; step < size - 1; ++step) {

int256 swapped = 0;

for (uint256 i = 0; i < size - step - 1; ++i) {

if (array[i] > array[i + 1]) {

int256 temp;

temp = array[i];

array[i] = array[i + 1];

array[i + 1] = temp;

swapped = 1;

}

}

if (swapped == 0) {

break;

}

}

return array;

}

}

Distinct Elements

Task

Create a function distinct(array, length of array) . This distinct() will take two arguments - a dynamic uint type array and length of the array.

The distinct() will return the number of distinct elements in an array. For Example - If array =[4, 2, 3, 4] then distinct() will return 3. If array =[1, 1, 1] then distinct() will return 1.

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day4 {

function distinct(int256[] memory array, uint256 len)

public

pure

returns (uint256)

{

uint256 i;

uint256 j;

uint256 count = 1;

for (i = 1; i < len; i++) {

for (j = 0; j < i; j++) {

if (array[i] == array[j]) {

break;

}

}

if (i == j) {

count++;

}

}

return count;

}

}

Search an element

Task

Create a function search(array, length of array,element to search) . This search() will take three arguments - a dynamic uint type array ,length of the array, element that we need to search.

The search() will search an element in an array.

If element is found then it will return 1 otherwise 0. For Example - If array =[4, 2, 3, 4] search element = 2 then search() will return 1. If array =[1, 2, 3] search element = 4 then search() will return 0.

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day5 {

function search(

int256[] memory arr,

uint256 size,

int256 toSearch

) public pure returns (uint256) {

for (uint256 i = 0; i < size; i++) {

if (arr[i] == toSearch) {

return 1; //if element is found

}

}

return 0; //if element is not found

}

}

Sum of array

Task

Create a function sumarray(array, length of array) . This sumarray() will take two arguments - a dynamic int type array and length of the array.

The sumarray() will add all elements of an array. For Example - If array =[4, 2, 3, 4] then sumarray() will return 13 i.e. 4+2+3+4 = 13 If array =[1, 2, 3] then sumarray() will return 6 i.e. 1+2+3 = 6

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day6 {

function sumarray(int256[] memory arr, uint256 size)

public

pure

returns (int256)

{

int256 sum = 0;

for (uint256 i = 0; i < size; i++) {

sum += arr[i];

}

return sum;

}

}

Expression Solving

Task

Find the sum of the series 1 + x + x^2+ x^3+ .. + x^n

Create a function expression(x,n) .

The expression() will find the sum of the above expression. For Example - If x =2 n = 3 then expression() will return 15. If x =5 n = 1 then expression() will return 6.

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day1 {

function expression(uint256 x, uint256 n) public pure returns (uint256) {

uint256 sm = 1;

uint256 m = 1;

for (uint256 i = 1; i <=n; i++) {

m = m \* x;

sm = sm + m;

}

return sm;

}

}

Second largest element

Task

Create a function secondMax(array, length of array) . This secondMax() will take two arguments - a dynamic int type array and length of the array.

The secondMax() will find the second largest element in an array. For Example - If array =[40, 12, 31, 6] then secondMax() will return 31. If array =[9, 122, 37, 54] then secondMax() will return 54.

Note - Function will be public.

You can do this :)

Ans—

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day2 {

function secondMax(int256[] memory arr, uint256 size) public pure returns(int256) {

for (uint256 i = 0; i < size - 1; i++) {

for (uint256 j = i + 1; j < size; j++) {

if (arr[i] < arr[j]) {

int256 temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

return arr[1];

}

}

HCF

Task HCF or Highest Common Factor is the greatest number which divides each of the two or more numbers.

Create a function hcf(num1,num2) . This hcf() will take two arguments uint type number1 and number2.

The hcf() will find the of number1 and number2. For Example - If num1 = 60 num2 = 75 then hcf() will return 15. If num1 = 31 and num2 = 23 then hcf() will return 1.

Note - Function will be public.

You can do this :)

Ans:-

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day3 {

function hcf(uint256 n1, uint256 n2) public pure returns (uint256) {

if (n2 != 0) {

return hcf(n2, n1 % n2);

} else {

return n1;

}

}

}

Books structure

Task Create a structure -> struct Book { string title; string author; uint256 book\_id; }

Create two functions a) setBook(title(string),author(string),id(uint)) - To set the book title, author and id given in the structure. b) getBook() - To get the book title,author and id.

Note - Function will be public.

You can do this :)

Ans;--

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day4 {

struct Book {

string title;

string author;

uint256 book\_id;

}

Book book;

function setBook(

string memory \_titile,

string memory \_author,

uint256 \_id

) public {

book = Book(\_titile, \_author, \_id);

}

function getBook()

public

view

returns (

string memory,

string memory,

uint256

)

{

return (book.title, book.author, book.book\_id);

}

}

Books array

Task Create a structure -> struct Book { string title; string author; uint256 book\_id; }

Create "Book type" dynamic array. This "Book type" array will contain different types of Book.

Create a counter( initialised by 1) which increases every time when a Book is added to the "Book type" array. This counter will act as a id for the Books added in the "Book type" array. Every Book in the "Book type" array will have a unique id.

Create two functions a) setBook(title(string),author(string),id(uint)) - To set the book title, author and id given in the "Book type" array. b) getBook(id(uint)) - To get the book title,author and id of the Book whoose id is passed as an argument.

Note - Function will be public.

You can do this :)

Ans:

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day5 {

struct Book {

string title;

string author;

uint256 book\_id;

}

Book[5] book;

uint256 i = 1;

function setBook(

string memory \_titile,

string memory \_author,

uint256 \_id

) public {

book[i] = Book(\_titile, \_author, \_id);

i++;

}

function getBook(uint256 \_i)

public

view

returns (

string memory,

string memory,

uint256

)

{

return (book[\_i].title, book[\_i].author, book[\_i].book\_id);

}

}

Array within Students

Task Create a structure -> struct Student { string name; uint256[3] marks; }

Create two functions a) set(name of student, marks in maths,marks in science,marks in english) - To set the name and marks of a student in marks array. b) get() - To get the name,marks in maths,marks in science and marks in english of the student. Note - Function will be public.

You can do this :)

Ans:--

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day6 {

struct Student {

string name;

uint256[3] marks;

}

Student student;

function set(

string memory \_name,

uint256 maths,

uint256 science,

uint256 english

) public {

uint256 i = 0;

student.name = \_name;

student.marks[i] = maths;

i++;

student.marks[i] = science;

i++;

student.marks[i] = english;

}

function get()

public

view

returns (

string memory,

uint256,

uint256,

uint256

)

{

return (

student.name,

student.marks[0],

student.marks[1],

student.marks[2]

);

}

}

Enum

Task Create an -> enum House { SMALL, MEDIUM, LARGE }

Create a variable of enum type.

Create a function setLarge(). This will set the value of the enum type variable (created at step 1) with LARGE.

Create a function getChoice() which will return the value of the enum type variable (created at step 1) .

Note - Function will be public.

You can do this :)

Ans:

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day1 {

enum House {

SMALL,

MEDIUM,

LARGE

}

House choice;

function setLarge() public {

choice = House.LARGE;

}

function getChoice() public view returns (House) {

return choice;

}

}

Onwer of the contract

Task

Create a constructor() to hold the address of the owner. [Hint: use msg.sender]

Create a function returnOwner() which returns the address of the owner. This returnOwner() can only be called by the owner (that we set using constructor()). If some other address tries to call this returnOwner() it should revert with this statement "You are not the owner" Note - Function will be public.

You can do this :)

Ans

// SPDX-License-Identifier: MIT

pragma solidity >=0.5.0 <0.9.0;

contract Day2 {

address owner;

constructor() {

owner = msg.sender;

}

function returnOwner() public returns (address) {

require(owner == msg.sender, "You are not the owner");

return owner;

}

}

Mapping

Task

Create a mapping(address => uint256).

Create a function updateBalance(balance to update) . This updateBalance() will take one argument - a uint type variable. This function will update the balance of the address who has called this updateBalance().

Create a function checkBalance(address of a user) . This checkBalance(address user)() will take one argument - a address type variable. This function will return the balance of the user who has called this function. With this function user can only check his/her own balance only. If the user tries to check some other address balance the checkBalance() should revert with "You are not the owner of the account".

For example - 0x123 address can check his own balance only. If 0x123 tries to check the balance of 0xab1 then the checkBalance() will revert with "You are not the owner of the account"

Note - Function will be public.

You can do this :)

Ans:

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.22 <0.9.0;

contract Day3 {

mapping(address => uint256) balances;

function updateBalance(uint256 newBalance) public {

balances[msg.sender] = newBalance;

}

function checkBalance(address \_user) public view returns (uint256) {

require(msg.sender == \_user, "You are not the owner of the account");

return balances[\_user];

}

}

Transfer Amount

Task

Create a constructor() to set the owner of contract . [Hint: use msg.sender]

Create a payable function send(address type array for storing addresses, uint type array to store the amount). This send() will do the following -

The send() can only be called by owner of the contract.

Check whehter the length of the address array is equal to length of the amount array. If not revert with "to must be same length as amount".

If the length of both array are equal. Then it will transfer the amount from the amount array to the address array in such a way address at ith index of address array receives the amount at ith index of amount array. For Example - If address array =[0x412, 0x2ab, 0x3de] and amount array = [10,5,90] then send() will transfer amount 10 to address 0x412 , amount 5 to address 0x2ab, amount 90 to address 03de.

Note - Function will be public.

You can do this :)

//SPDX-License-Identifier: UNLICENSED

pragma solidity >=0.5.0 <0.9.0;

contract Day4 {

address owner;

constructor() {

owner = msg.sender;

}

function send(address payable[] memory to, uint256[] memory amount)

public

payable

ownerOnly

{

require(to.length == amount.length, "to must be same length as amount");

for (uint256 i = 0; i < to.length; i++) {

to[i].transfer(amount[i]); //to array - 0x00 0x01 0x02

//amount array - 10 20 30

}

}

modifier ownerOnly() {

require(msg.sender == owner);

\_;

}

}

Team Players

Objective-

To create an array which stores addresses of the team players.

Task-

Create the following variables and functions -

Create an array(address type) which stores only 16 elements.

getTeamPlayers() - To return the entire array elements.

selectJerseyNumber() - It will take only one argument of uint type and returns the players address from the array created above. a) The argument passed into the function must be greater than equal to zero and less than equal to 15. [Hint - Use require]

You can do this :-)

Ans:-

//SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.5.0 <0.9.0;

contract Day5 {

address[16] team;

function getTeamPlayers() public view returns (address[16] memory) {

return team;

}

function selectJerseyNumber(uint256 playerId) public returns (uint256) {

require(playerId >= 0 && playerId <= 15);

team[playerId] = msg.sender;

return playerId;

}

}

String

Objective-

To conactenate two strings.

Task-

Create the following function -

1)concatenate ( ) - This function will take two arguments i.e string 1 and string 2. It will concatenate string 1 and string 2 and will return the concatenated string.

You can do this :-)

Ans:-

//SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.5.0 <0.9.0;

contract Day6 {

function length(string memory str) public pure returns (uint256) {

bytes memory str\_bytes = bytes(str);

return str\_bytes.length;

}

function concatenate(string memory str1, string memory str2)

public

pure

returns (string memory)

{

bytes memory str\_bytes1 = bytes(str1);

bytes memory str\_bytes2 = bytes(str2);

string memory str = new string(str\_bytes1.length + str\_bytes2.length);

bytes memory str\_bytes = bytes(str);

uint256 k = 0;

for (uint256 i = 0; i < str\_bytes1.length; i++) {

str\_bytes[k] = str\_bytes1[i];

k++;

}

for (uint256 i = 0; i < str\_bytes2.length; i++) {

str\_bytes[k] = str\_bytes2[i];

k++;

}

return string(str\_bytes);

}

}