1번 문제 제출완료

Complete the given source code to build a "Monte-Carlo Simulation" program explained in pages 25, 26, and 27 of the lecture slide "[07] Functions and Modular Programming" to get an approximate value of the π.

입/출력 예시

␣ : 공백 ↵ : 줄바꿈 ⇆ : 탭

예시 1

입력

```
1000
```

출력

```
Monte-Carlo␣PI␣=␣3.08000000000000007105,␣(Number␣of␣Sample␣=␣1000)
```

※ 입출력 형식을 잘 지켜주세요

2번 문제 제출완료

Complete the function csv_get_value() that returns the next integer value from the CSV string that is passed to the argument "str". That is, whenever it is called, it returns an integer in the CSV string one by one, sequentially from the beginning.

If there is no more value to read, return -1 to stand for the "end".

No global variable is allowed to use in implementing your function.

3번 문제 제출완료

Complete hexa2decimal() in the given source code that returns a decimal integer value corresponding to a hexa string given by the parameter "str_hexa". You can safely assume that all hexa string starts with "0x" and only 0~9 and A~F (Capital/Upper) characters are used.

실행 화면 예시

␣ : 공백 ↵ : 줄바꿈 ⇆ : 탭

```
0x1F34␣=␣7988
```

4번 문제 제출완료

Read the reference on printf().

Write proper format specifiers for the printf() functions in lines 16, 17, and 18 in the given source code to get the execution results shown in the example.

- line 16 printf() : left-justified
- line 17 printf() : start with 0x, Hexa(0~9, A~F), minimum field width = 4, leading 0s
- line 18 printf() : start with 0o, Octa(0~7), minimum field width = 4, leading 0s

입/출력 예시

␣ : 공백   ↵ : 줄바꿈   ⇆ : 탭

예시 1

입력

```
2
```

출력

```
+----------------------+↵
|123456|123456|12345678|↵
+----------------------+↵
|61875␣|0xF1B3|0o170663|↵
|57634␣|0xE122|0o160442|↵
|45473␣|0xB1A1|0o130641|↵
|45275␣|0xB0DB|0o130333|↵
|32106␣|0x7D6A|0o076552|↵
|4053␣␣|0x0FD5|0o007725|↵
|37779␣|0x9393|0o111623|↵
|20652␣|0x50AC|0o050254|↵
|59995␣|0xEA5B|0o165133|↵
|2712␣␣|0x0A98|0o005230|↵
+----------------------+↵
```

※ 입출력 형식을 잘 지켜주세요

5번 문제 제출완료

[Game of Stone 1]

Two players called Alice and Bob are playing a game with a number of stones. Alice always plays first, and the two players move in alternating turns. The game's rules are as follows:

- In a single move, a player can remove either 2, 3, or 5 stones from the game board.
- If a player is unable to make a move because there are not enough stones, that player loses the game.

For example, let the starting number of stones *n = 4*, they can make the following moves:

- Alice removes 2 stones leaving 2, Bob will then remove 2 stones and win
- Alice removes 3 stones and leaves 1 stone. Bob cannot move and loses.

So, Alice would make the second play and win the game.

Complete the function Alice_Move() in the given source code, **which returns the largest number of stones to remove in Alice's turn if she can win.** Otherwise, that is, **If she can't win, return 2 (the smallest number of stones that she can remove). You can safely assume that Alice_Move() is called only when she can remove stones.**

(Alice가 제거할 돌의 수를 결정하는 함수 Alice_Move()를 작성하라. 이 함수는 Alice가 이길 수 있는 상황에서 이기는 상황을 계속 유지시키는 최대 값을 반환한다. Alice가 이길 수 없는 상황의 경우 제거할 수 있는 최소 값인 2를 반환하여야 한다. 참고로 Alice_Move()는 항상 돌을 제거할 수 있는 상황에서 호출된다고 가정한다.)

The number of current stones remaining is maintained in the global variable **gn_stones** which is accessible inside Alice_Move(). You can add new functions if necessary. But you are not allowed to modify other functions nor to add additional global variables.

입/출력 예시

␣ : 공백 ↵ : 줄바꿈 ⇆ : 탭

예시 1

입력

```
20
```

출력

```
The␣starting␣number␣of␣stones␣20↵
Alice>␣5␣removed␣:␣[20␣=>␣15␣stones␣remained]↵
Bob>␣3␣removed␣:␣[15␣=>␣12␣stones␣remained]↵
Alice>␣5␣removed␣:␣[12␣=>␣7␣stones␣remained]↵
Bob>␣2␣removed␣:␣[7␣=>␣5␣stones␣remained]↵
Alice>␣5␣removed␣:␣[5␣=>␣0␣stones␣remained]↵
Congratulations␣Alice,␣You␣Win␣!!!
```

예시 2

입력

```
65
```

출력

```
The starting number of stones 65
Alice> 2 removed : [65 => 63 stones remained]
Bob> 5 removed : [63 => 58 stones remained]
Alice> 2 removed : [58 => 56 stones remained]
Bob> 5 removed : [56 => 51 stones remained]
Alice> 2 removed : [51 => 49 stones remained]
Bob> 5 removed : [49 => 44 stones remained]
Alice> 2 removed : [44 => 42 stones remained]
Bob> 5 removed : [42 => 37 stones remained]
Alice> 2 removed : [37 => 35 stones remained]
Bob> 5 removed : [35 => 30 stones remained]
Alice> 2 removed : [30 => 28 stones remained]
Bob> 2 removed : [28 => 26 stones remained]
Alice> 5 removed : [26 => 21 stones remained]
Bob> 3 removed : [21 => 18 stones remained]
Alice> 3 removed : [18 => 15 stones remained]
Bob> 3 removed : [15 => 12 stones remained]
Alice> 5 removed : [12 => 7 stones remained]
Bob> 5 removed : [7 => 2 stones remained]
Alice> 2 removed : [2 => 0 stones remained]
Congratulations Alice, You Win !!!
```

※ 입출력 형식을 잘 지켜주세요

6번 문제 제출완료

Recall the "[06] Finding Square Root ver.2 - Bisection Search" introduced before, which finds an approximation to the square root of any non-negative number x.

While we have applied an "Exhaustive Search" or "Bisection Search" to get an answer, the most commonly used approximation algorithm for this problem is Newton's method. In this lab, we are going to implement Newton's method and want to compare its performance with the previous ones.

Refer to the Internet to understand Newton's method; For example, visit the Wikipedia page.

Complete the findroot_newton() function in the given source code that implements "Newton's method" to find an approximation to the square root of any non-negative number.

입/출력 예시

␣ : 공백 ↵ : 줄바꿈 ⭾ : 탭

예시 1

입력

```
25↵
```

출력

```
#␣of␣Guesses␣=␣4999900↵
[EXHAUSTIVE]␣sqrt(25.000000)␣~=␣4.99990000034497139580↵
#␣of␣Guesses␣=␣15↵
[BISECTION␣]␣sqrt(25.000000)␣~=␣4.99992370605468750000↵
#␣of␣Guesses␣=␣4↵
[␣␣NEWTON␣␣]␣sqrt(25.000000)␣~=␣5.00001295304868431657↵
[␣␣MATH.H␣␣]␣sqrt(25.000000)␣~=␣5.00000000000000000000↵
```

※ 입출력 형식을 잘 지켜주세요