5.x — Chapter 5 comprehensive quiz

BY ALEX ON JULY 16TH, 2015 | LAST MODIFIED BY NASCARDRIVER ON JANUARY 24TH, 2020

Quick review

If statements allow us to execute a statement based on whether some condition is true. *Else statements* execute if the associated if statement is false. You can chain together multiple if and else statements.

Switch statements provide a cleaner and faster method for selecting between a number of discrete items. Switch statements pair great with enumerations.

Goto statements allow the program to jump to somewhere else in the code. Don't use these.

While loops allow the program to loop as long as a given condition is true. The condition is evaluated before the loop executes.

Do while loops are the same as while loops, but the condition is evaluated after the loop execution. They're great for menus or things that need to execute at least once.

For loops are the most used loop, and are perfect when you need to loop a specific number of times.

Break statements allow us to break out of a `switch`, `while`, `do while`, or `for` loop. Or a range-based `for` loop, which we haven't covered yet.

Continue statements allow us to move immediately to the next loop iteration. Be careful when using these with while and do while loops, as your loop counter may not get incremented properly.

And finally, random numbers give us a way to make our programs behave different each time they are run. We'll see an example of this in the quiz below!

Quiz time!

Warning: The quizzes start getting harder from this point forward, but you can do it. Let's rock these quizzes!

1) In the chapter 2 comprehensive quiz, we wrote a program to simulate a ball falling off of a tower. Because we didn't have loops yet, the ball could only fall for 5 seconds.

Take the program below and modify it so that the ball falls for as many seconds as needed until it reaches the ground.

In constants.h:

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

namespace myConstants
{
    constexpr double gravity{ 9.8 }; // in meters/second squared
}
#endif
```

In your main code file:

```
#include "constants.h"

#include <iostream>

// gets initial height from user and returns it
```

```
double getInitialHeight()
{
    std::cout << "Enter the height of the tower in meters: ";
    double initialHeight;
    std::cin >> initialHeight;
    return initialHeight;
}

// Returns height from ground after "secondsPassed" seconds
double calculateHeight(double initialHeight, int secondsPassed)
{
    // Using formula: [ s = u * t + (a * t^2) / 2 ], here u(initial velocity) = 0
    double distanceFallen{ (myConstants::gravity * secondsPassed * secondsPassed) / 2.0 };
    double currentHeight{  initialHeight - distanceFallen };

    return currentHeight;
}

// Prints height every second till ball has reached the ground
void printHeight(double height, int secondsPassed)
{
    if (height > 0.0)
    {
        std::cout << "At " << secondsPassed << " seconds, the ball is at height:\t" << height
<<
            " meters\n";
    }
    else
        std::cout << "At " << secondsPassed << " seconds, the ball is on the ground.\n";
}

void calculateAndPrintHeight(double initialHeight, int secondsPassed)
{
    double height{ calculateHeight(initialHeight, secondsPassed) };
    printHeight(height, secondsPassed);
}

int main()
{
    const double initialHeight{ getInitialHeight() };

    calculateAndPrintHeight(initialHeight, 0);
    calculateAndPrintHeight(initialHeight, 1);
    calculateAndPrintHeight(initialHeight, 2);
    calculateAndPrintHeight(initialHeight, 3);
    calculateAndPrintHeight(initialHeight, 4);
    calculateAndPrintHeight(initialHeight, 5);

    return 0;
}
```

Show Solution

2a) Implement a game of hi-lo. First, your program should pick a random integer between 1 and 100. The user is given 7 tries to guess the number.

If the user does not guess the correct number, the program should tell them whether they guessed too high or too low. If the user guesses the right number, the program should tell them they won. If they run out of guesses, the program should tell them they lost, and what the correct number is. At the end of the game, the user should be asked if they want to play again. If the user doesn't enter 'y' or 'n', ask them again.

Note: You do not need to implement error handling for the user's guess.

Here's what your output should look like:

```
Let's play a game.  I'm thinking of a number.  You have 7 tries to guess what it is.
Guess #1: 64
Your guess is too high.
Guess #2: 32
Your guess is too low.
Guess #3: 54
Your guess is too high.
Guess #4: 51
Correct! You win!
Would you like to play again (y/n)? y
Let's play a game.  I'm thinking of a number.  You have 7 tries to guess what it is.
Guess #1: 64
Your guess is too high.
Guess #2: 32
Your guess is too low.
Guess #3: 54
Your guess is too high.
Guess #4: 51
Your guess is too high.
Guess #5: 36
Your guess is too low.
Guess #6: 45
Your guess is too low.
Guess #7: 48
Your guess is too low.
Sorry, you lose.  The correct number was 49.
Would you like to play again (y/n)? q
Would you like to play again (y/n)? f
Would you like to play again (y/n)? n
Thank you for playing.
```

Hints:
* If your compiler is C++11 capable, use the Mersenne Twister algorithm from chapter **5.9 -- Random number generation** to pick a random number.
* If your compiler is not C++11 capable, you can use rand() (also presented in chapter **5.9 -- Random number generation**) to pick a random number
* Write a function that allows the user to play a single game of hi-lo.
* Write a function that asks the user if they want to play again and handles the looping logic for an incorrect input.

Show Solution

2b) Update your previous solution to handle invalid input (e.g. 'x') or valid input with extraneous characters (e.g. "43x") when the user is guessing a number.

Hint: Write a separate function to handle the user inputting their guess (along with the associated error handling).

Show Solution

6.1 -- Arrays (Part I)

Index

5.11 -- Introduction to testing your code

C++ TUTORIAL | PRINT THIS POST

681 comments to 5.x — Chapter 5 comprehensive quiz

« Older Comments 1 ... 7 8 9

Fabio Henrique
January 24, 2020 at 4:08 am · Reply

I tried to fix every issue that you pointed
constants.hpp:

```
#ifndef CONSTANTS_HPP
#define CONSTANTS_HPP
#include <ios>
#include <limits>
namespace constants {
inline static constexpr auto maxstreamsize{
    std::numeric_limits<std::streamsize>::max()};
}
#endif
```

main.cpp

```
#include "constants.hpp" // constants::maxstreamsize
#include "random.hpp" //effolkronium::random_static
#include <iostream>

const void gamePresentation() {
  std::cout << "Hello, and welcome to the guessing game !\n";
}

const void gameExplanation() {
  std::cout << "You have 7 chances to guess a number that is between 1 and "
               "100.\nGood luck !\n";
}

bool isInRange(const int guess) {
  if (guess >= 1 && guess <= 100) {
    return true;
  } else {
    std::cout << "Out of range !\nThe guess must be a number between "
                 "1 and 100\n";
    return false;
  }
}

const void clearInput() {
  if (std::cin.fail())
    std::cin.clear();
```

```
  std::cin.ignore(constants::maxstreamsize, '\n');
}

int askForGuess(int count) {
  int guess{};
  while (true) {
    std::cout << "Guess #" << count << ": ";
    std::cin >> guess;
    clearInput();
    if (isInRange(guess))
      return guess;
    else
      continue;
  }
  return guess;
}

const void guessRight() {
  std::cout << "You guessed the right number !\nCongratulations and thanks for "
               "playing!\n";
}

const void printRnd(const int rnd) {
  std::cout << "The correct number was: " << rnd << '\n';
}

const void guessWrong() {
  std::cout << "Well, it seems you didn't guessed the right number this time :/"
               "\nBest luck on the next try !\nThanks for playing !\n";
}
const void outOfGuesses() {
  std::cout << "Uh oh, it seems that you're out of out of guesses :/\n";
}

bool playAgain() {
  char playagain{};
  do {
    std::cout << "Would you want to play again ?\n(y/n) : ";
    std::cin >> playagain;
    clearInput();
  } while (playagain != 'y' && playagain != 'n');
  return (playagain == 'y');
}

int main() {
  // assume the user didn't guessed the number to output the right exit
  // message
  bool guessedwrong{true};
  // creating rnd before main loop to use it in the exit message if user
  // don't guess the number in the last try
  int rnd{};

  // separate gamePresentation of gameExplanation to only output the first in
  // the beginning
  gamePresentation();
  do {
    gameExplanation();
    rnd = effolkronium::random_static::get(1, 100);
    std::cout << rnd << '\n'; // remove this!!!
    // creating count before to output outOfGuesses message if user ran out of
    // guesses
    int count{0};
```

```
89      do {
90        int guess{askForGuess(++count)};
91        if (guess < rnd) {
92          std::cout << "Your guess is too low...\n";
93          guessedwrong = true;
94          continue;
95        } else if (guess > rnd) {
96          std::cout << "Your guess is too high...\n";
97          guessedwrong = true;
98          continue;
99        } else {
100         std::cout << "You won !!!\n";
101         guessedwrong = false;
102         break;
103       }
104     } while (count < 7);
105     if (count == 7)
106       outOfGuesses();
107     if (playAgain())
108       continue;
109     else
110       break;
111   } while (true);
112   if (guessedwrong) {
113     guessWrong();
114     printRnd(rnd);
115   } else
116     guessRight();
117   return 0;
118 }
```

Just some questions:
-Is it good or bad to have a const function like my clearInput() ? it seems to work pretty fine as a const.
-And I was trying to remember why did I disallowed 0 and allowed it again after, and it is because of the cin.ignore(), every time it runs on a bad input it transforms the value of guess back to 0 AFAICS, like when a char is input in the guesses... my question is, if we wanted to allow 0 in our gueses, how could we do that and still be able to ignore extraneous input with cin.ignore() ?
-I just declared maxstreamsize as a long because clangd told me that it would evaluate for a long, it can evaluate for something different in some case ?

Thanks

PS: and I knew you probably would react like that, but this formatting is not disgusting for me XD

nascardriver
January 24, 2020 at 4:26 am · Reply

You still have magic numbers/strings (Line 10, 11, 15, 19, 84, 104, 105).
Line 107-111: `while (playAgain());`

> Is it good or bad to have a const function like my clearInput() ?
That function returns `void`, you can't assign to a `void` anyway, so the `const` doesn't do anything. For other types, it can catch very rare mistakes, but it prevents an important optimization which you'll learn about later (Move sematics). `const` return values do more bad than good (Unless you're returning a reference/pointer (Covered later)).

> [std::cin] transforms the value of guess back to 0 AFAICS
That's correct. You should use `std::cin.fail()` to check if extraction failed, not the value.

> clangd told me that it would evaluate for a long
Your editor/ide/tool tells you what the type is AFTER preprocessing files. Take this simple example

```
#if defined(HAS_BIG_POWERFUL_PC)
using streamsize = long long;
#else
using streamsize = signed char;
#endif
```

If you have a big powerful pc, `streamsize` will be `long long`, and clangd will tell you that it is `long long`. But if someone uses your code on another machine, `streamsize` is a `char`. If you have a look at a **reference** (A lesson about cppreference is in the making), you'll see that `std::streamsize` is "a signed integral type", but not one type in particular. That's all you can rely on. If your implementation uses a `long`, that doesn't mean every implementation uses a `long`.

Fabio Henrique
January 24, 2020 at 6:23 am · Reply

I think I understand everything now, I'm sorry if sometimes I take some time to understand but it's because English is not my main language so I have to strive to understand sometimes.

Now, in constants.hpp:

```
#ifndef CONSTANTS_HPP
#define CONSTANTS_HPP
#include <ios>
#include <limits>
namespace constants {
inline static constexpr auto maxstreamsize{
    std::numeric_limits<std::streamsize>::max()};
inline static constexpr int minguessvalue{1};
inline static constexpr int maxguessvalue{100};
inline static constexpr int maxchances{7};
} // namespace constants
#endif
```

In main.cpp:

```
// constants::maxstreamsize, maxguessvalue, minguessvalue, maxchances
#include "constants.hpp"
#include "random.hpp" //effolkronium::random_static
#include <iostream>

void gamePresentation() {
  std::cout << "Hello, and welcome to the guessing game !\n";
}

void gameExplanation() {
  std::cout << "You have " << constants::maxchances
            << " chances to guess a number that is between "
            << constants::minguessvalue << " and " << constants::maxguessvalue
            << ".\nGood luck !\n";
}

bool isInRange(const int guess) {
  if (guess >= constants::minguessvalue && guess <= constants::maxguessvalue) {
    return true;
  } else {
    std::cout << "Out of range !\nThe guess must be a number between "
              << constants::minguessvalue << " and " << constants::maxguessvalu
e
              << '\n';
    return false;
```

```
  }
}

void clearInput() {
  if (std::cin.fail())
    std::cin.clear();
  std::cin.ignore(constants::maxstreamsize, '\n');
}

int askForGuess(int count) {
  int guess{};
  do {
    std::cout << "Guess #" << count << ": ";
    std::cin >> guess;
    clearInput();
    if (isInRange(guess))
      return guess;
    else
      continue;
  } while (true);
  return guess;
}

void guessRight() {
  std::cout << "You guessed the right number !\nCongratulations and thanks for
 "
               "playing!\n";
}

void printRnd(const int rnd) {
  std::cout << "The correct number was: " << rnd << '\n';
}

void guessWrong() {
  std::cout << "Well, it seems you didn't guessed the right number this time
 :/"
               "\nBest luck on the next try !\nThanks for playing !\n";
}
void outOfGuesses() {
  std::cout << "Uh oh, it seems that you're out of out of guesses :/\n";
}

bool playAgain() {
  char playagain{};
  do {
    std::cout << "Would you want to play again ?\n(y/n) : ";
    std::cin >> playagain;
    clearInput();
  } while (playagain != 'y' && playagain != 'n');
  return (playagain == 'y');
}

int main() {
  // assume the user didn't guessed the number to output the right exit
  // message
  bool guessedwrong{true};
  // creating rnd before main loop to use it in the exit message if user
  // don't guess the number in the last try
  int rnd{};

  // separate gamePresentation of gameExplanation to only output the first in
  // the beginning
```

```
  gamePresentation();
  do {
    gameExplanation();
    rnd = effolkronium::random_static::get(constants::minguessvalue,
                                           constants::maxguessvalue);
    std::cout << rnd << '\n'; // remove this!!!
    // creating count before to output outOfGuesses message if user ran out of
    // guesses
    int count{0};
    do {
      int guess{askForGuess(++count)};
      if (guess < rnd) {
        std::cout << "Your guess is too low...\n";
        guessedwrong = true;
        continue;
      } else if (guess > rnd) {
        std::cout << "Your guess is too high...\n";
        guessedwrong = true;
        continue;
      } else {
        std::cout << "You won !!!\n";
        guessedwrong = false;
        break;
      }
    } while (count < constants::maxchances);
    if (count == constants::maxchances)
      outOfGuesses();
  } while (playAgain());
  if (guessedwrong) {
    guessWrong();
    printRnd(rnd);
  } else
    guessRight();
  return 0;
}
```

Now just a last question, I understand when you say "use consts when working with magick numbers", but what you mean when you say "use consts when working with strings" ?! Is that something like 'magick numbers can be strings too' ?! or I'm misunderstanding ?!
And I have to start searching more on the references before making these types of questions, thank you for this !

nascardriver
January 24, 2020 at 6:31 am · Reply

Your `inline`s don't do anything, because the variables are `static` (`inline`: Can be used in multiple files. `static`: Only visible in this file).

Code looks good now :)

We say "magic number", but it can be anything. If two things have the same meaning, they should be the same named thing.

Fabio Henrique
January 24, 2020 at 9:38 am · Reply

oh okay, I got that..
Magic Numbers can be anything actually, is just a way of naming. Got it!
and I didn't knew that the compiler would let me define inline and static to the same

variable if they are opposites without even issuing a warning or something like that... I'm gonna change just to static then !

Fabio Henrique
January 23, 2020 at 7:10 am · Reply

to the first question of the quiz, I came out with this solution, and actually I was pretty surprised that it is pretty different than the one you presented here:

in constants.h:

untouched

in main.cpp:

```
#include "constants.h"

#include <iostream>

// gets initial height from user and returns it
double getInitialHeight()
{
    std::cout << "Enter the height of the tower in meters: ";
    double initialHeight;
    std::cin >> initialHeight;
    return initialHeight;
}

// Returns height from ground after "secondsPassed" seconds
double calculateHeight(double initialHeight, int secondsPassed)
{
    // Using formula: [ s = u * t + (a * t^2) / 2 ], here u(initial velocity) = 0
    double distanceFallen{ (myConstants::gravity * secondsPassed * secondsPassed) / 2.0 };
    double currentHeight{  initialHeight - distanceFallen };

    return currentHeight;
}

// Prints height every second till ball has reached the ground
void printHeight(double height, int secondsPassed)
{
    if (height > 0.0)
    {
        std::cout << "At " << secondsPassed << " seconds, the ball is at height:\t" << hei
ght <<
            " meters\n";
    }
    else
        std::cout << "At " << secondsPassed << " seconds, the ball is on the ground.\n";
}

void calculateAndPrintHeight(double initialHeight)
{
    double height{ initialHeight };
    for(int secondsPassed{};  height > 0.0 ; ++secondsPassed )
    {
        height = calculateHeight(initialHeight , secondsPassed);
        printHeight(height, secondsPassed);
    }
}

```

```
47  int main()
48  {
49      const double initialHeight{ getInitialHeight() };
50
51      calculateAndPrintHeight(initialHeight);
52
53      return 0;
    }
```

I saw just now that was said here to another person on the comments section that he modified calculateHeight() and because of that it became not reusable, in mine I tried not to touch any function but the calculateAndPrintHeight() one, since it doesn't have a return that can modify the behavior of the program or of the other functions on it, everything seems to still be reusable AFAICS. Let me know if this solution is good or bad please.
Thank you!

nascardriver
January 23, 2020 at 7:17 am · Reply

The reason the code became non-reusable was because the calculation was performed inside of `calculateAndPrintHeight`. In your code, that's not the case. You have a separate `calculateHeight` and `printHeight` function. The height can be calculated or printed without doing the other, good job!
Your formatting is inconsistent, consider using your editor's auto-formatter.

Fabio Henrique
January 23, 2020 at 8:35 am · Reply

thank you ! is very nice to know that I'm going the right way :D
and I don't format my code to post here because it does things that was said here before that were not the style you guys would use in these tutorial series, I'm using NeoVim with Clangd and the formatting defaults to LLVM style, and this does some things like the opening curly brace should always be on the same line that the last statement not on a new line, and the closing curly brace of an if should always be on the same line that the else if it exists, not on a new line, and etc. I prefer this style but in favor of the style you guys prefer I don't format my code to post here. If you think it's ok if I do let me know please

nascardriver
January 23, 2020 at 8:43 am · Reply

It doesn't matter how you format your code as long as it's consistent, which yours isn't. clang-format can be configured to place braces wherever you like. Even if you can't get it to put the braces where you want, consistency is more important than following some suggestion.

Fabio Henrique
January 24, 2020 at 2:20 am · Reply

yes, I know it's configurable, actually I already have it configured but it's pretty different than the style that you guys use here, that's what I was trying to say, I was posting the code here trying to follow the code style that you guys like before formatting, actually I always format my code, I already tried other styles but I really like my code compact. And yes, consistence is more important, I'm going to always format my

code to post here then.. beginning with my guessing game:
in constants.hpp

```
#ifndef CONSTANTS_HPP
#define CONSTANTS_HPP
#include <ios>
#include <limits>
namespace constants {
inline static constexpr long maxstreamsize{
    std::numeric_limits<std::streamsize>::max()};
}
#endif
```

in main.cpp

```
#include "constants.hpp" // constants::maxstreamsize
#include "random.hpp" //effolkronium::random_static
#include <iostream> //std::cin, std::cout

void gamePresentation() {
  std::cout << "Hello, and welcome to the guessing game !\n";
}

void gameExplanation() {
  std::cout << "You have 7 chances to guess a number that is between 0
and "
               "100.\nGood luck !\n";
}

bool isInRange(int guess) {
  if (guess >= 0 && guess <= 100) {
    return true;
  } else {
    std::cout << "Number out of range !\nThe guess must be a number be
tween "
                 "0 and 100\n";
    return false;
  }
}

void clearInput() {
  if (std::cin.fail())
    std::cin.clear();
  std::cin.ignore(constants::maxstreamsize, '\n');
}

int askForGuess(int count) {
  int guess{};
  while (true) {
    std::cout << "Guess #" << count << ": ";
    std::cin >> guess;
    clearInput();
    if (!guess) {
      std::cout << "Invalid guess !\n";
      continue;
    }
    if (isInRange(guess))
      return guess;
    else
      continue;
  }
  return guess;
}
```

```
49
50  void guessRight() {
51    std::cout << "You guessed the right number !\nCongratulations and th
52  anks for "
53                 "playing!\n";
54  }
55
56  void printRnd(int rnd) {
57    std::cout << "The correct number was: " << rnd << '\n';
58  }
59
60  void guessWrong() {
61    std::cout << "Well, it seems you didn't guessed the right number thi
62  s time :/"
63                 "\nBest luck on the next try !\nThanks for playing !\n"
64  ;
65  }
66  void outOfGuesses() {
67    std::cout << "Uh oh, it seems that you're out of out of guesses :/
68  \n";
69  }
70
71  bool playAgain() {
72    char playagain{};
73    do {
74      std::cout << "Would you want to play again ?\n(y/n) : ";
75      std::cin >> playagain;
76      clearInput();
77    } while (playagain != 'y' && playagain != 'n');
78    (playagain == 'y') ? playagain = true : playagain = false;
79    return playagain;
80  }
81
82  int main() {
83    bool playing{true};
84    // assume the user didn't guessed the number to output the right exi
85  t
86    // message
87    bool guessedwrong{true};
88    // creating rnd before main loop to use it in the exit message if us
89  er
90    // don't guess the number in the last try
91    int rnd{};
92
93    // separate gamePresentation of gameExplanation to only output the f
94  irst in
95    // the beginning
96    gamePresentation();
97    while (playing) {
98      gameExplanation();
99      rnd = effolkronium::random_static::get(0, 100);
100     // creating count before to output outOfGuesses message if user ra
101 n out of
102     // guesses
103     int count{0};
104     for (; count < 7;) {
105       int guess{askForGuess(++count)};
106       if (guess < rnd) {
107         std::cout << "Your guess is too low...\n";
108         guessedwrong = true;
109         continue;
110       } else if (guess > rnd) {
```

```
111             std::cout << "Your guess is too high...\n";
112             guessedwrong = true;
113             continue;
114           } else {
115             std::cout << "You won !!!\n";
116             guessedwrong = false;
117             break;
118           }
119         }
120         if (count == 7)
              outOfGuesses();
            playing = playAgain();
          }
          if (guessedwrong) {
            guessWrong();
            printRnd(rnd);
          } else
            guessRight();
          return 0;
        }
```

This is the final version of it.
It looks good or bad in your opinion ?!
And I did a version using the mersenne twister to practice already, this one with the effolkronium header is just easier and more good looking IMO. And I learned a little bit more after taking a look inside the header code too so I decided to use it.
Let me know if I'm missing something or if I can improve something please. Thank you !!

nascardriver
January 24, 2020 at 2:44 am · Reply

> It looks good or bad in your opinion ?!
Oh that formatting is absolutely disgusting, but if you like it, keep it. Some people like smarts, some people like jeeps, you do you.

To the code,
- `std::numeric_limits::max()` returns a `std::streamsize`, not a `long` (It may be a `long`, but it doesn't have to). The type is already in the call, you can use `auto`.
- Magic numbers/strings, use constants.
- Line 36: You're disallowing 0 (`!guess` is the same as `guess == 0`), but you're allowing 0 in line 15.
- Good separation into many functions.
- Line 72: Use the conditional operator only if you use its result. Don't mix types. `playagain` is a `char`, not a `bool`. `return (playagain == 'y')`;
- Line 88: The first time this is encountered, it's `true`. Use a do-while-loop instead (That also gets rid of `playing`).
- You're using a for-loop as if it were a while-loop. Use a while-loop or move `++count` into the loop's header.

Eric
January 18, 2020 at 4:18 pm · Reply

In your answer key to question 2 (your code lines 26 - 37 shown below listed as 1-12), why does the line
36 return (ch == 'y');
not always cause the function to return ch = 'y'.

I tested and see that your program works correctly, but to my understanding, since it is after the do loop, it should always happen.
What am I misunderstanding?
P.S. Thank you for undergoing this site. I have been enjoying learning cpp.

```
1  bool playAgain()
2  {
3      // Keep asking the user if they want to play again until they pick y or n.
4      char ch;
5      do
6      {
7          std::cout << "Would you like to play again (y/n)? ";
8          std::cin >> ch;
9      } while (ch != 'y' && ch != 'n');
10     return (ch == 'y');
11 }
```

nascardriver
January 19, 2020 at 1:13 am · Reply

`ch` is modified inside of the loop (Line 8). The loop stops when `ch` is either 'y' or 'n' (Line 9). In line 10, `ch` is 'y' or 'n'. If it's 'y', `playAgain` returns `true`, otherwise `false`.

Eric
January 20, 2020 at 8:24 am · Reply

Thank you. I understand this better now.

« Older Comments 1 ... 7 8 9