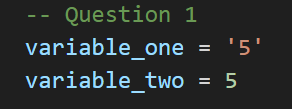
**Types and Logic Quiz**

Q1



*Which of the following answer is correct?*

Opt1: They are both numbers

Feedback if chosen: Strings are identified by 'the quotation marks' or sometimes "these"

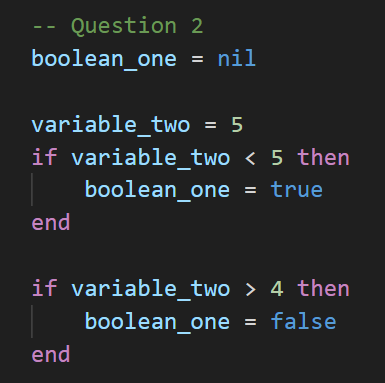
Opt2: variable\_one is a number  
variable\_two is a string

Feedback: Strings are identified by 'the quotation marks' or sometimes "these"

**Opt3 (correct): variable\_one is a string**

variable\_two is a number

Q2



*After these lines of code, what will the value of boolean\_one be?*

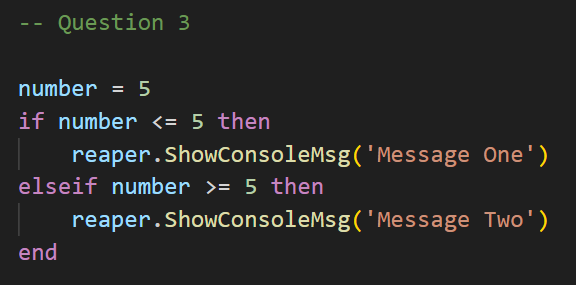
Opt1: nil

**Opt2 (correct): false**

Feedback: That's right. when the code checks if variable\_two is great than 4, this check will be true, so the part of the code where it sets boolean\_one to be false will run

Opt3: true

Q3



*Which message will be printed to the console. <= means 'less than or equal to' and >= means 'greater than or equal to*

**Opt1 (correct): 'Message One'**

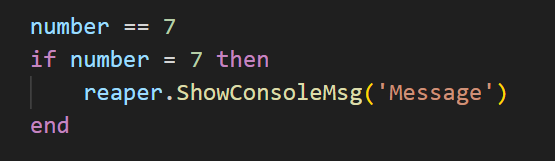
Feedback: Yes that is right. Both of these conditional statements are true, however the way the system works is that it evaluates the if statements from top to bottom. Since the first one is true, it enters that area of the code but then skips the subsequence elseif statements

Opt2: 'Message Two'

Feedback: Both of these conditional statements are true, however the way the system works is that it evaluates the if statements from top to bottom. Since the first one is true, it enters that area of the code but then skips the subsequence elseif statements, so Message One will be the only one that is printed

**Follow Up Quiz**

Q1



*What will happen in this section of code?*

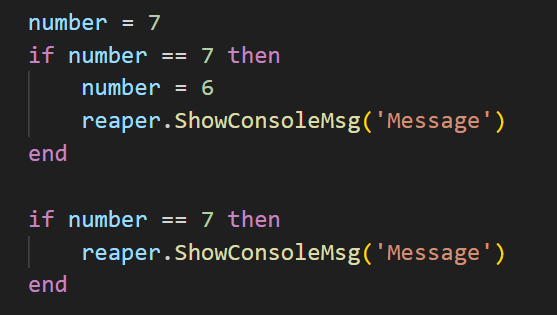
**Opt1 (correct): We will get an error**

Feedback: That's correct. Remember that == is an equivalency operator and the single = is an assignment operator. Putting if before an assignment will create an error

Opt2: 'Message' will be printed to the console

Feedback: Remember that == is an equivalency operator and the single = is an assignment operator. Putting if before an assignment will create an error

Q2



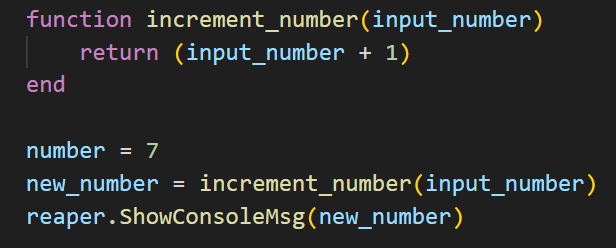
*What will we see on the console after all of this?*

Opt1: MessageMessage

Feedback: No. Because, after the first if statement, we update the value of number, the second if statement will not proceed to the second print

Opt2 (correct): Message

Q3



*What will happen in this code?*

Opt1: There will be no errors, but nothing will actually happen

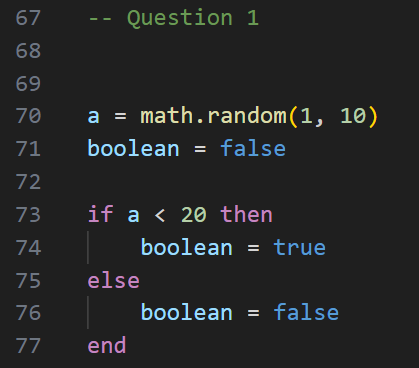
**Opt2 (correct): 8 will be printed to the console**

Feedback: That's right. We start off with a value of 7 in the variable named 'number.' Then, new\_number is initialised with a value of 8. This is what prints to the console

Opt3: 7 will be printed to the console

**Random Numbers Quiz**

Q1



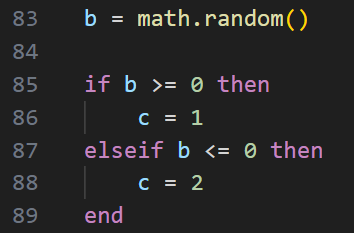
*After this code runs, what will be the value of 'boolean'?*

**Opt1 (correct): true**

Feedback: That's right, it must be true because a will be a number between 1 and 10 so it must be always less than 20

Opt2: false

Q2



*After this code runs what will the value of c be?*

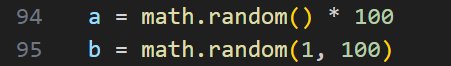
*Note that >= means 'greater than or equal to' and <= means 'less than or equal to'*

**Opt1 (correct): c = 1**

Feedback: That's right. math.random() without anything in the brackets will give a random number between 0 and 1, including decimal places. So b will always be either 0 or greater than 0

Opt2: c = 2

Q3



*What is the best description of the possible outcomes of a and b?*

Opt1: a and b will both have the same possibilities of outcomes

Feedback: No, not quite. math.random(1, 100) can only be whole numbers (integers) between 1 and 100, whereas a will be anything from 0 to 100 including decimal place values

**Opt2 (correct): a will be any decimal place numbers (float) between 0 and 100**

b will be any whole number (integer) between 1 and 100

Q4



*What is the best description of the possible outcomes of a?*

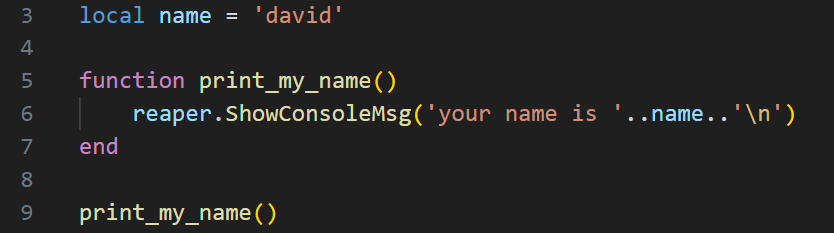
Opt1: A will be any whole number (integer) between -10 and 10

**Opt2 (correct) : A will be any number, including decimal places between -10 and 10**

Feedback: That's right. You'll see a pattern here, when we want the random numbers to include decimal places, we start with math.random(), which gives a value between 0 and 1, and then we scale up or down and move the range around by multiplying/dividing or adding/subtracting, like we've done here

**Variable Scope Quiz**

Q1



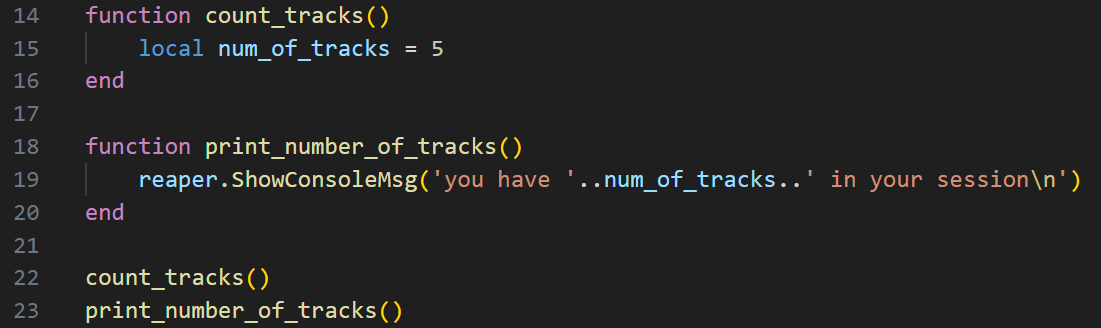
*If we ran this code, what would you expect the outcome to be?*

Opt1: It would error, because name is not in scope at line 6

**Opt2 (correct): a console message: "your name is david"**

Feedback: Because name is initialised at line 3, and it is not bounded by anything, it is available anywhere in this file

Q2



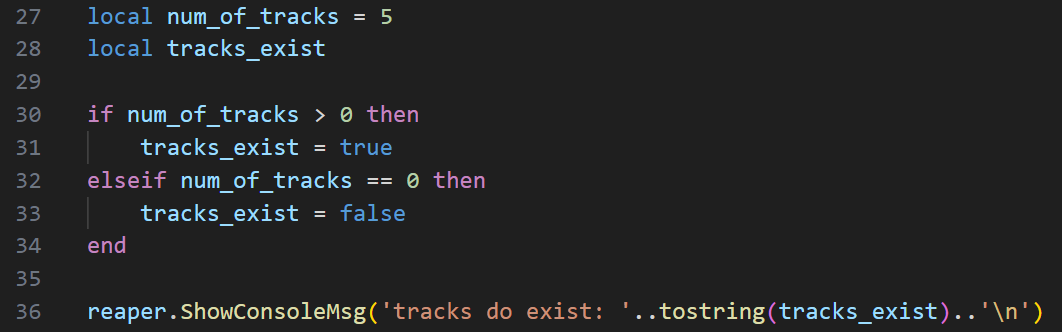
*If we ran this code, what would the outcome be?*

**Opt1 (correct): Error for trying to concatenate a nil variable at line 19**

Feedback: Yes, if you look at line 15, where num\_of\_tracks get initialised, it is only available in the scope of the function count\_tracks(), so trying to use it at line 19 will not work

Opt2: a console message: "you have 5 in your session"

Q3



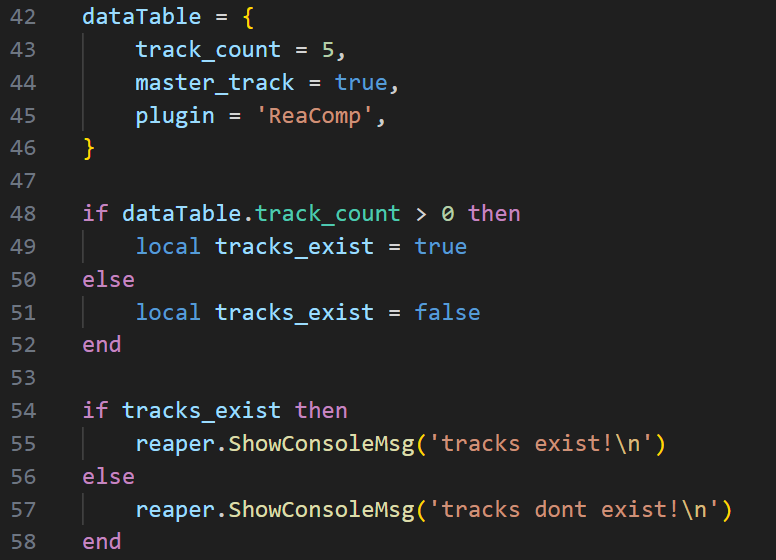
*If we run this code, what would the outcome be?*

**Opt1 (correct): console message: "tracks do exist: true"**

Feedback: That's right. Our local variables initialised at lines 27 and 28 are available throughout this excerpt of code

Opt2: Error. tracks\_exist is not in scope at line 36

Q4

**

*What would we expect to see if we run this code?*

Opt1: Error. The variable tracks\_exist is not in scope at line 54

Feedback: This is true! But we actually wouldn't get an error

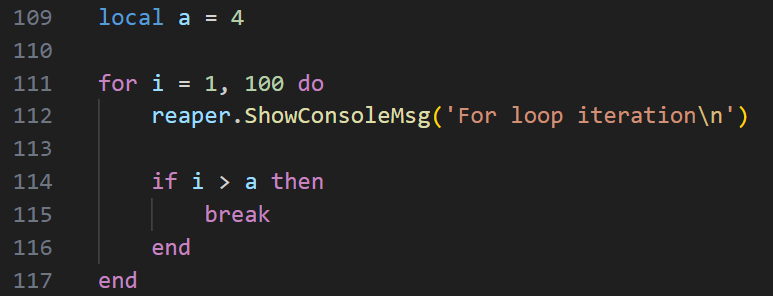
Opt2: console message: "tracks exist!"

**Opt3 (correct): console message: "tracks dont exist!"**

Feedback: That's right. This is a bit of a trick question. At line 49, the variable will be initialised as true, but because it is local it immediately goes out of scope after the if statement. At line 54, tracks\_exist will be nil, so we would enter the 'else' portion of the code at line 57

**Loops Quiz**

Q1



*How many times do we expect 'For loop iteration' to print to the console?*

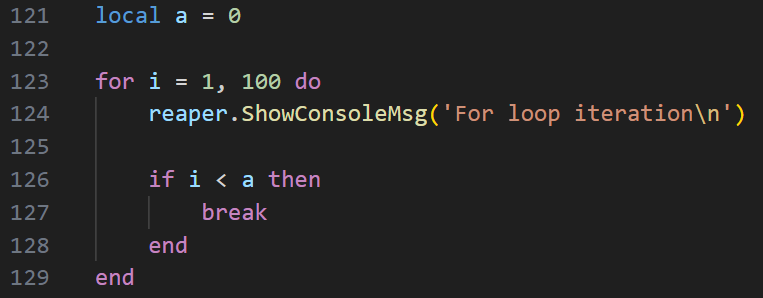
Opt1: 100

Opt2: 99

**Opt3 (correct): 5**

Feedback: That's right. The break keyword stops the loop prematurely based on the condition

Q2



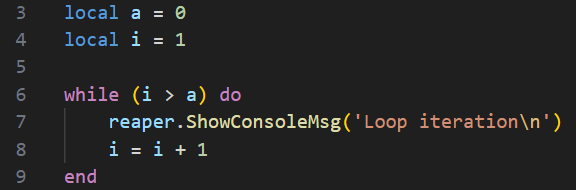
*How many times do we expect "For loop iteration" to print to the console?*

**Opt1 (correct): 100**

Opt2: It will print infinitely until it crashes because i is never less than a

Feedback: Tricked you! This might be the case if it were a 'while loop', but this is a for loop, so it won't be infinite

Q3



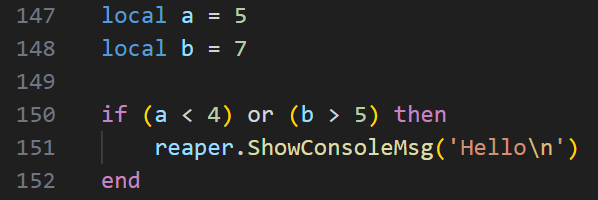
*How many times will 'Loop iteration' print out?*

Opt1: Only once

**Opt2 (correct): Infinitely many times until Reaper crashes**

Feedback: Yes that's right. The condition i > a will never be true, so it will loop around and around until Reaper runs out of memory

**Boolean Logic Quiz**

Q1  
****

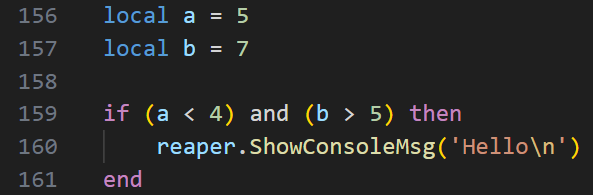
*Will 'Hello' print?*

**Opt1 (correct): Yes**

Feedback: There are two conditions in the 'if' statement. Because they are separated by the 'or' if at least one of them is true, then Hello will print. b &gt; 5 is true

Opt2: No

Q2

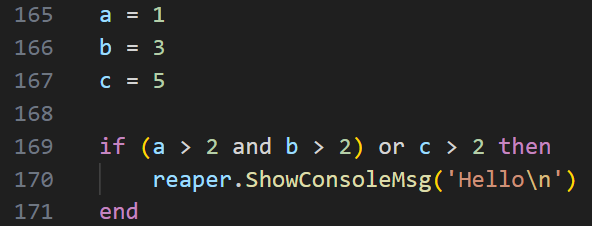


*Will 'hello' print?*

Opt1: Yes

**Opt2 (correct): No**

Feedback: There are two conditions inside the 'if' statement. But this time they are separated by the 'and'. This means that BOTH must be true in order for the whole thing to be true. Because a < 4 is false, the whole thing is false, and 'hello' doesn't print

Q3  
  


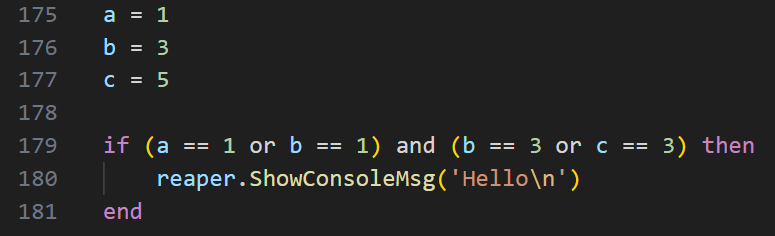
*Will 'hello' print?*

**Opt1 (correct): Yes**

Feedback: I read it like this. There are two broad conditions: (a > 2 and b > 2) being the first, and (c > 2) being the second. Because they are separated by the 'or' only one of those need to be true. The first is false, but the c > 2 is true. So the whole thing will be true

Opt2: No

Q4



*Will 'hello' print?*

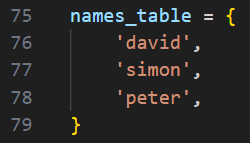
**Opt1 (correct): Yes**

Feedback: for the whole thing to be true, (a == 1 or b == 1) must be true as well as (b == 3 or c == 3). Inside of each bracket is the 'or' word, and the inside of each bracket has one true condition. So the whole thing is true

Opt2: No

**Table Indexes Quiz**

Q1



*How do we access the element 'peter' using its index?*

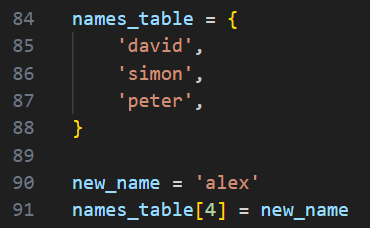
**Opt1(correct): names\_table[3]**

Opt2: names\_table[2]

Feedback: In other languages it might be 2, but in Lua, the indexes start from 1, not 0

Opt3: names\_table.3

Q2



*What is a description of what is happening above?*

Opt1: An error. Line 91 isn't allowed

**Opt2 (correct): A fourth element is being added to names\_table. Its index/key is 4 and its value is 'alex'**

Feedback: That's right. This will add a new entry 'alex' at index 4

Q3  


*For this example, let's say we don't know how many entries there are in this table as we have been adding them during runtime.*

*However, you do know that 'alex' is the LAST element in the table. How would we access the last element?*

**Opt1 (correct): names\_table[#names\_table]**

Feedback: That's right. #names\_table is the number of elements in the table. And since indexes start at 1 in lua, #names\_table is the same number as the last element in the table!

Opt2: You can't without knowing how many elements there are

Q4



*This is the same example as the previous question; we don't know how many elements there are in the table.*

*Your task is to select an element from names\_table at random.*

*How would we do this?*

Opt1: names\_table[math.random()]

Feedback: No, math.random() without any arguments returns a random value between 0 and 1. Only whole numbers can be indexes in the table

**Opt2 (correct): names\_table[math.random(1, #names\_table)]**

Feedback: Nice! So breaking it down from the inside out. math.random(1, #names\_table) will return a random whole number (integer) between 1 and whatever the last index is. We then use that outcome as the index of the element we randomly chose to access