https://www.udemy.com/learn-how-to-code/?couponCode=INNOVATE

1. Initialize a SLICE of int using a composite literal; print out the slice; range over the slice printing out just the index; range over the slice printing out both the index and the value
    a. solution
2. Initialize a MAP using a composite literal where the key is a string and the value is an int; print out the map; range over the map printing out just the key; range over the map printing out both the key and the value
    a. solution
3. Create a new type called person which is STRUCT that stores fName and lName
    a. solution
4. Using the STRUCT "person", using a composite literal create a value of type person and assign it to a variable with the identifier "p1"; print out "p1"; print out just the field fName for "p1"
    a. solution
5. Take the STRUCT "person" in the previous exercise and add a field called "favFood" that stores a slice of string; for the variable "p1" use a composite literal to add values to the field favFood; print out the values in favFood; also print out the values for "favFood" by using a for range loop
    a. solution
6. Add a method to type "person" with the identifier "walk". Have the func return this string: <person's first name> is walking. Remember, you make a func a method by giving the func a receiver.

```
func (r receiver) identifier(parameters) (returns) {
        <code>
}
```

    To return a string, use fmt.Sprintln. Call the method assigning the returned string to a variable with the identifier "s". Print out "s".
    a. solution
7. Create a new type: vehicle. The underlying type is a struct. The fields: doors, color. Create two new types: truck & sedan. The underlying type of each of these new types is a struct. Embed the "vehicle" type in both truck & sedan. Give truck the field "fourWheel" which will be set to bool. Give sedan the field "luxury" which will be set to bool.
    a. solution
8. Using the vehicle, truck, and sedan structs: using a composite literal, create a value of type truck and assign values to the fields; using a composite literal, create a value of type sedan and assign values to the fields. Print out each of these values. Print out a single field from each of these values.
    a. solution
9. Give a method to both the "truck" and "sedan" types with the following signature
        transportationDevice() string
    Have each func return a string saying what they do. Create a value of type truck and

populate the fields. Create a value of type sedan and populate the fields. Call the method for each value.

   a. solution

10. Create a new type called “transportation”. The underlying type of this new type is interface. An interface defines functionality. Said another way, an interface defines behavior. For this interface, any other type that has a method with this signature …

        transportationDevice() string

    … will automatically, implicitly implement this interface. Create a func called “report” that takes a value of type “transportation” as an argument. The func should print the string returned by “transportationDevice()” being called for any type that implements the “transportation” interface. Call “report” passing in a value of type truck. Call “report” passing in a value of type sedan.
    a. solution
11. Create a new type called “gator”. The underlying type of “gator” is an int. Using var, declare an identifier “g1” as type gator (var g1 gator). Assign a value to “g1”. Print out “g1”. Print the type of “g1” using fmt.Printf(“%T\n”, g1)
    a. solution
12. Adding onto this code: Using var, declare an identifier “x” as type int (var x int). Print out “x”. Print the type of “x” using fmt.Printf(“%T\n”, x)
    a. solution
13. Adding onto this code: Can you assign “g1” to “x”? Why or why not?
    a. solution
14. Adding onto this code: You will now learn about CONVERSION. This is called “CASTING” in a lot of other languages. Since “g1” is of type “gator” but its underlying type is an “int”, we can use “CONVERSION” to convert the value to an int. Here is how you do it:
    a. solution
15. Now add a method to type gator with this signature ...

        greeting()

    … and have it print “Hello, I am a gator”. Create a value of type gator. Call the greeting() method from that value.
    a. solution
16. Adding onto this code: create another type called “flamingo”. Make the underlying type of “flamingo” bool. Give “flamingo” a method with this signature …

        greeting()

    … and have it print “Hello, I am pink and beautiful and wonderful.” Now create a new type “swampCreature”. The underlying type of “swapCreature” is interface. The behavior which the “swampCreature” interface defines is such that any type which has this method …

        greeting()

    … will implicitly implement the “swampCreature” interface. Create a func called “bayou” which takes a value of type “swampCreature” as an argument. Have this func print out the greeting for whatever “swampCreature” might be passed in.

a. solution

17. Using the short declaration operator, create a variable with the identifier “s” and assign “i'm sorry dave i can't do that” to “s”.
    i. Print “s”.
    ii. Print “s” converted to a slice of byte.
    iii. Print “s” converted to a slice of byte and then converted back to a string.
    iv. Using slicing, print just “i’m sorry dave”
    v. Using slicing, print just “dave i can’t”
    vi. Using slicing, print just “can’t do that”
    vii. print every letter of “s” with one rune (character) on each line
    b. solution

Good job on working through the exercises. Here is your reward:

https://www.youtube.com/watch?v=otCpCn0l4Wo

#1
CREATE TYPE PERSON
THE UNDERLYING TYPE IS A STRUCT
FIELDS: FIRST, LAST, AGE

#2
USE A COMPOSITE LITERAL
TO CREATE A VALUE OF TYPE PERSON
AND ASSIGN IT TO A VARIABLE
USING THE SHORT DECLARATION OPERATOR