CS118
Variables

Variable
Computer programming (*n*): A *variable* is a name for a memory location

Hardcode (*v*): To place a fixed value into a computer program. A value in a computer program is *not* considered hardcoded if it was either computed by the program; or came from outside the program, frequently through a user interface.

For each of the following, provide a separate Python3 program that starts with the command `print('\x1b[2J')` then accomplishes the tasks specified. Submit all .py files together as a single ZIP file.
For the italicized questions, provide a comment in your code in answer to that question.

Defining
1. Create a variable named `intvar` that is hardcoded with an integer, displaying the value of the variable when the program is run by putting the variable name inside the parentheses after a `print` statement. *Describe what is actually occurring inside the computer when this line of code is run but do not describe it in terms of variables; rather, describe it in terms of memory, transistors, bits, etc.*

```
# Clear the console
print("\x1b[2J")

intvar = 25

print(intvar)

# The transistors that make up the memory location referred to as "intvar" are being adjusted to represent the number 25.
```

2. Create a variable named `floatvar` that is hardcoded with a floating point value, displaying the value of the variable when the program is run by `print`ing the variable.

```
# Clear the console
print("\x1b[2J")

floatvar = 12.3

print(floatvar)
```

3. In programming, a "string" is a sequence of characters (i.e. symbols). There is no meaning to this sequence except as a set of characters – so any digits contained within this sequence (even when are ONLY digits present) are not treated as quantities. For example, the sequence "123" is simply the sequence of the digit 1, the digit 2, and the digit 3 – not the value one hundred twenty-three. Strings are "delimited" (i.e. marked off) by double-quotes (that is one character, not two single-quotes) before the first character and after the last character.

Create a variable named `strvar` that is hardcoded with a string of your choosing. Display the value of the variable when the program is run by `print`ing the variable. *What happens if you use no quotes when hardcoding your string?*

```
# Clear the console
print("\x1b[2J")

strvar = "This is my string"

print(strvar)

# If no quotes are used, an error occurs because the Python interpreter is attempting to use the characters that makeup the string like a variable, and that/
those variables don't exist in the program.
```

Mathematics and Assignment
4. Hardcode the variable `addend` with an integer that is less than 50. Hardcode the variable `my_number` with another integer that is also less than 50. Compute the value of the variable `my_answer` as the sum of `addend` and `my_number`, displaying its value by `print`ing the variable.

```
# Clear the console
print("\x1b[2J")

addend = 23

my_number = 24

my_answer = addend + my_number

print(my_answer)
```

5. Make a copy of #4. Change the variable `my_answer` to `my answer` (replace the underscore with a space). *Does the program still run? Obviously, replacing the underscore with a space caused an error – why do you think that is?*

```
# Clear the console
print("\x1b[2J")

addend = 23

my_number = 24

my answer = addend + my_number

print(my_answer)

# No, the program does not still run.  Replacing the underscore with a space caused Python to interpret the left-side of the assignment as two variables.
Assignment is not defined in that manner, so Python could not execute that command.
```

6. Make a copy of #4. Reverse the code line so that `my_answer` is on the right side of the equal sign and the code originally on the right side is now on the left side of the equal sign. *Does the program still run? Why do we usually refer to the equal sign as the "assignment operator"?*

```
# Clear the console
print("\x1b[2J")

addend = 23

my_number = 24

addend + my_number = my_answer

print(my_answer)

# No, the program does not run.

# The equal sign is the assignment operator because the value to the right is "assigned" the memory location (variable name) provided on the left.  The
assignment semantics (meaning of the operation) expects a single variable on the left.
```

Data Types

7. Hardcode the variable `addend` with an integer that is less than 50. Hardcode the variable `my_string` with a string that is no more than 10 characters long. Compute the value of the variable `my_answer` as the sum of `addend` and `my_string`. *What does the resulting error message mean? (Translate it into words that a common person would understand.)*

```
# Clear the console
print("\x1b[2J")

addend = 22

my_string = "My String"

my_answer = addend + my_string

# TypeError: unsupported operand type(s) for +: 'int' and 'str'

# This means the "+" operator does not work when the two operands are an integer and a string, respectively.

# Translation: Addition requires two numbers, not a number and a string
```

Swapping

8. Hardcode the variable `item1` with an integer that is less than 50. Hardcode the variable `item2` with a different integer less than 50. Swap the two values between the variables: Create a variable `temp` by assigning the variable `item1` to it. This saves the value that is in `item1` by copying it to `temp`. Next, assign the variable `item2` to the variable `item1` – this saves the value in `item2` in the variable `item1`. Finally, assign the variable `temp` to `item2`. After performing the swap, display `item1` and `item2` by using separate `print` statements to see that they have indeed swapped their values.

Swapping always takes these three fundamental steps:

```
DEFINE temp as item1
DEFINE item1 as item2
DEFINE item2 as temp
```

```
# Clear the console
print("\x1b[2J")

item1 = 22
item2 = 25
temp = item1
item1 = item2
item2 = temp

print(item1)
print(item2)
```