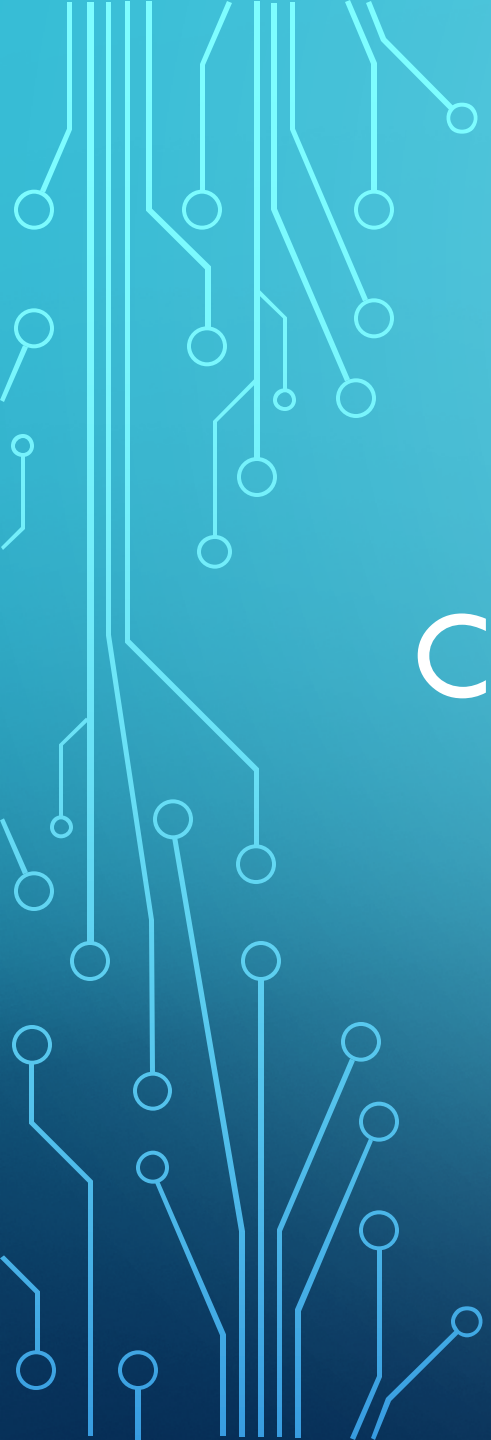


CSHARP – 4 - PROGRAM FLOW

OUTLINE

- Assignment 2 Review
- Boxing and Unboxing
- Flow control
- Lab

4.1 ASSIGNMENT REVIEW

- Program Structure
- Types
  - Value Types
  - Reference Types: String, Class
  - Conversions
  - Declaration
  - Assignment
- Flows

WHY DO WE NEED BOXING AND UNBOXING

- For example, the old collection type ArrayList only eats objects. That is, it only stores references to somethings that live somewhere. Without boxing you cannot put an int into such a collection. But with boxing, you can.
  - (Now, in the days of generics you don't really need this)
  - https://stackoverflow.com/questions/2111857/why-do-we-need-boxing-and-unboxing-in-c

- Test and Interview

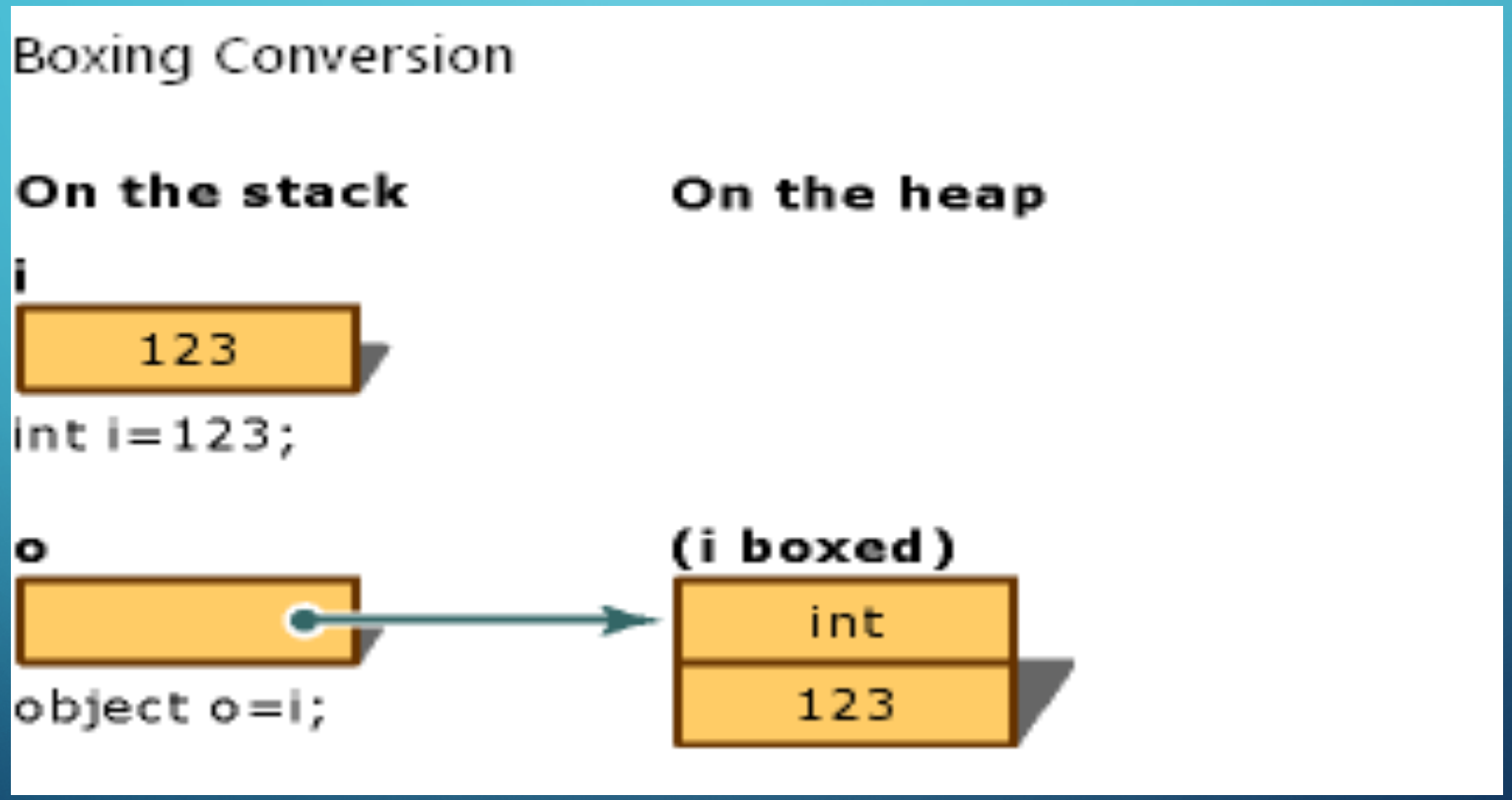
4.2 BOXING AND UNBOXING

- Boxing is the process of converting a value type to the type object or to any interface type implemented **by this value type (Structure).**
- When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap. Unboxing extracts the value type from the object. (Microsoft)

```
int i = 123;
object o = i;  // Implicit boxing
i = 456;       // Change the contents of i
Console.WriteLine("The value-type value = {0}", i);
Console.WriteLine("The object-type value = {0}", o);
```

(Run it in VS; Explanation in next slide)

MEMORY ALLOCATION IN BOXING

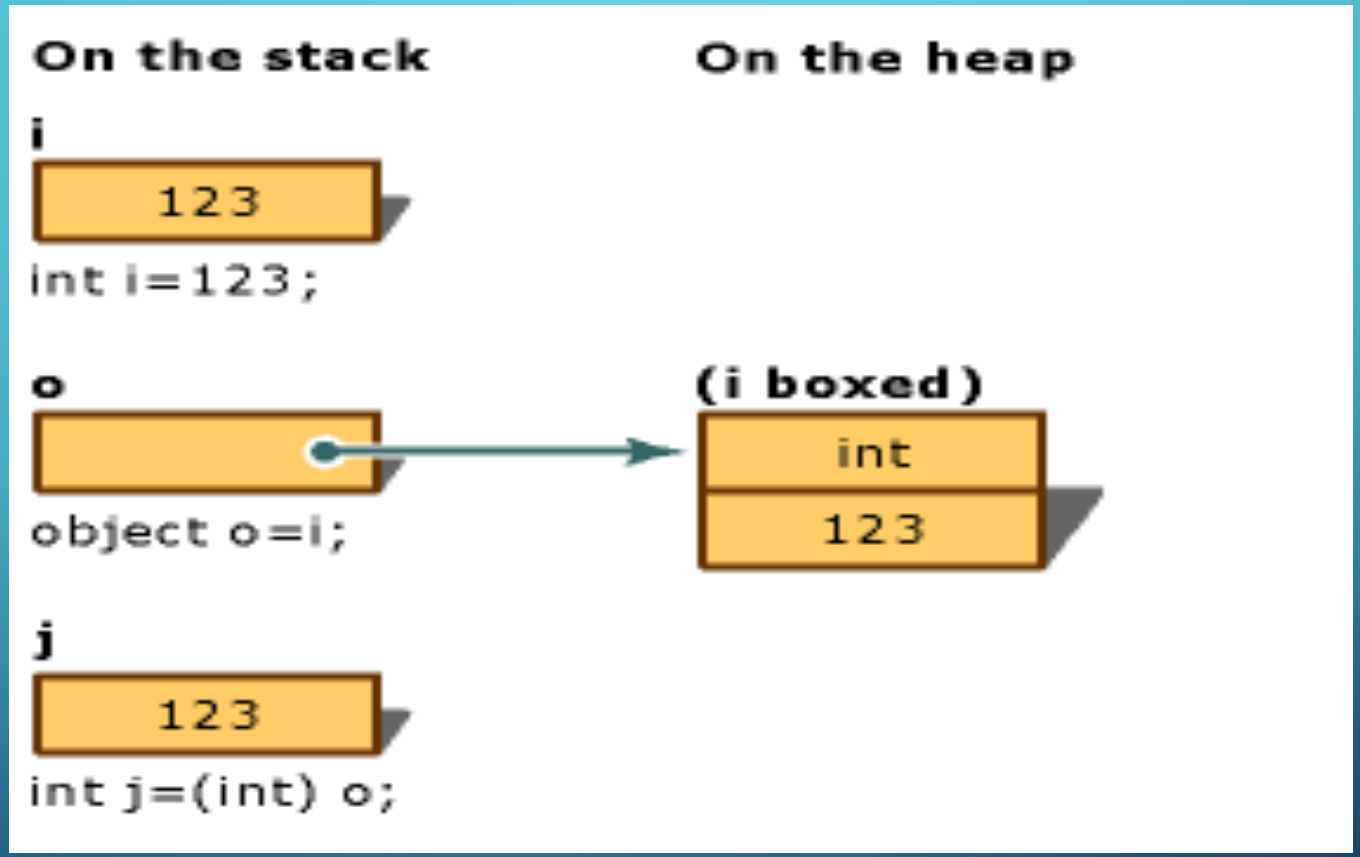


QUIRKS OF UNBOXING

```
int i = 123;        // a value type
object o = i;       // boxing
int j = (int)o;   // unboxing
```

must be the same type as the one boxed

MEMORY ALLOCATION OF UNBOXING



BOXING PERFORMANCE

In relation to simple assignments, boxing and unboxing are computationally expensive processes.

When a value type is boxed, a new object must be allocated and constructed. To a lesser degree, the cast required for unboxing is also expensive computationally.

UNBOXING (CONT'D)

Unboxing is an explicit conversion from the type object to a value type or from an interface type to a value type that implements the interface. An unboxing operation consists of:

- Checking the object instance to make sure that it is a boxed value of the given value type.
- Copying the value from the instance into the value-type variable.

UNBOXING (CONT'D)

For the unboxing of value types to succeed at run time, the item being unboxed must be a reference to an object that was previously created by boxing an instance of that value type.

Attempting to unbox null causes a NullReferenceException.

Attempting to unbox a reference to an incompatible value type causes an InvalidCastException.

AS OPERATOR

Using the as operator differs from a cast in C# in three important ways:

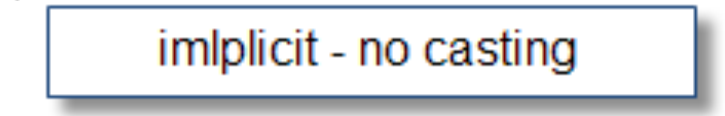
- It returns null when the variable you are trying to convert is not of the requested type or in it's inheritance chain, instead of throwing an exception.
- It can **only be applied to reference type variables converting to reference types.**
- Using as will not perform user-defined conversions, such as implicit or explicit conversion operators, which casting syntax will do.

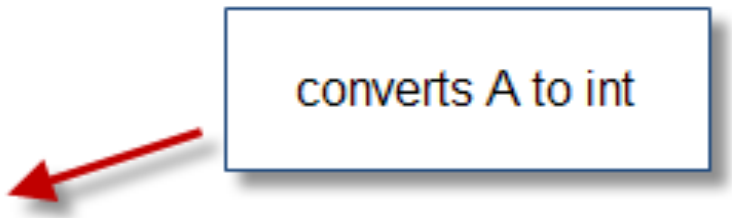
- The as operator is used to tell the application "I want you to try and convert this. It might not, and I know this, so don't throw an exception. I'll deal with it accordingly."

IMPLICIT CONVERSION OPERATOR

```
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = new A();
            Console.WriteLine(x.ToString());
        }
    }
    class A
    {
        int value = 0;
        public static implicit operator int(A a)
        {
            return a.value;
        }
    }

}
```





EXPLICIT CONVERSION OPERATOR

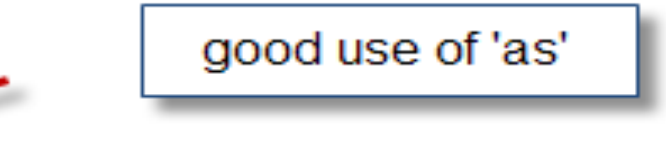
```
class Program
{
    static void Main(string[] args)
    {
        int x = (int)new A();
        Console.WriteLine(x.ToString());
    }
}
class A
{
    int value = 0;
    public static explicit operator int(A a)
    {
        return a.value;
    }
}
```

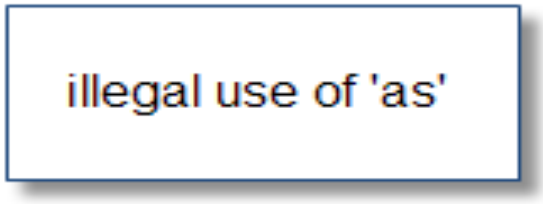
AS EXAMPLE

```
class Program
{
    static void Main(string[] args)
    {
        B b = new B();

        A a = b as A;

        int y = a as int;
    }
}
class A
{
    int value = 0;
    public static explicit operator int(A a)
    {
        return a.value;
    }
}
class B : A
{
}
```





- Running the codes will be helpful to understand boxing and unboxing.

4.3 FLOW CONTROL

- Operators
- Decision Making
- Error handling

OPERATORS

- Arithmetic Operators
- Relational Operators
- **Logical Operators**
- Bitwise Operators
- Assignment Operators
- Misc Operators

ARITHMETIC OPERATORS

+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	**Increment operator increases integer value by one**	**A++ = 11**
--	**Decrement operator decreases integer value by one**	**A-- = 9**

A++ VS ++A

- Codes

RELATIONAL OPERATORS

==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B)
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B)
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B)
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	A < B
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B)
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B)

LOGICAL OPERATORS

&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B)
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B)
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B)

BITWISE OPERATORS

p	q	p & q	p | q	p ^ q in one operand but not both
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

AN INTERVIEW QUESTION

- Swapping two integer variables without an intermediary variable
- Normal codes:

  int i = 3; int j = 4; int x = i; i= j; j=x;

- Bitwise version- Lab
  - A = A^B // A is now XOR of A and B
  - B = A^B // B is now the original A
  - A = A^B // A is now the original B

- Learn more when you need it

ASSIGNMENT OPERATORS

=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B assigns value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
……		

MISCELLANEOUS OPERATORS

sizeof()	Returns the size of a data type.	sizeof(int), returns 4.
typeof()	Returns the type of a class.	typeof(StreamReader);
&	Returns the address of an variable.	&a; returns actual address of the variable.
*	Pointer to a variable.	*a; creates pointer named 'a' to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y
is	Determines whether an object is of a certain type.	If(Ford is Car) // checks if Ford is an object of the Car class.
as	Cast without raising an exception if the cast fails.	Object obj = new StringReader("Hello");StringReader r = obj as StringReader;

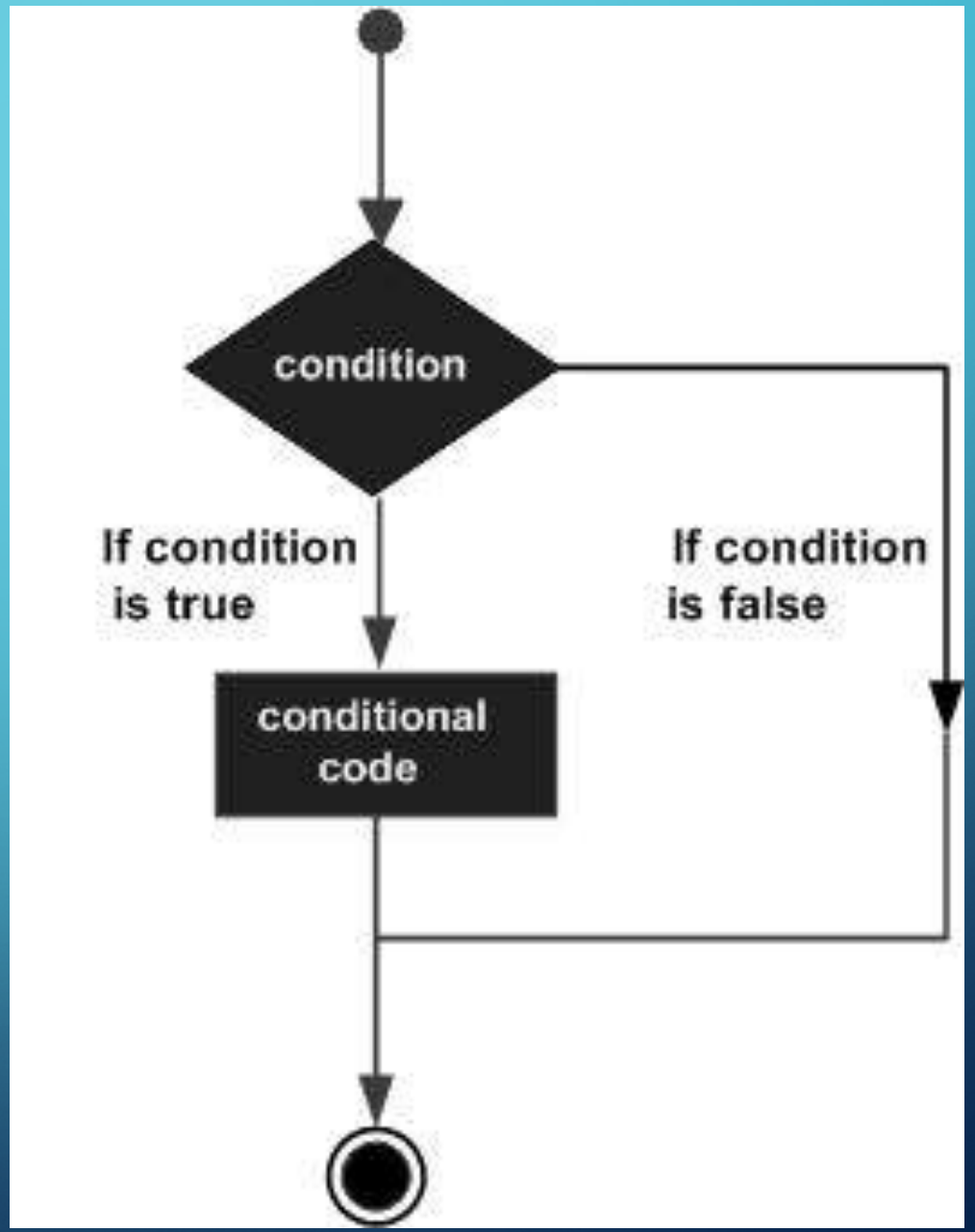
??

```
// Set y to the value of x if x is NOT null; otherwise,
// if x == null, set y to -1.
int y = x ?? -1;
```

NULL-CONDITIONAL OPERATORS (C# 6.0)

- var length = customers?.Substring(2)

DECISION MAKING



1	**if statement**An **if statement** consists of a boolean expression followed by one or more statements.
2	**if...else statement**An **if statement** can be followed by an optional **else statement,** which executes when the boolean expression is false.
3	**nested if statements**You can use one **if** or **else if** statement inside another **if** or **else if**statement(s).
4	**switch statement**A **switch** statement allows a variable to be tested for equality against a list of values.
5	**nested switch statements**You can use one **switch** statement inside another **switch**statement(s).

STUDY SOME SPECIAL CASES

Lab:

```
int s = 6;
 if (s==6)
 {              Console.Write("==6");          }
 else if (s>2)
 {              Console.Write("s>9");          }
 else
 {              Console.WriteLine("3rd line fo conditions");          }
```

A TYPICAL TEST QUESTION

C# switch Statement with grouped cases

```
int caseSwitch = 1;
switch (caseSwitch)
{
case 1: //break;
case 2: Console.WriteLine("Case 2");
break;
default:
Console.WriteLine("Default case");
break;
}
```

What will you get?

- In C# 6, the match expression must be an expression that returns a value of the following types:
  - a char.
  - a string.
  - a bool.
  - an integral value, such as an int or a long.
  - an enum value.
- Starting with C# 7, the match expression can be any non-null expression.

LAB

- Show dynamic welcome messages with

  good morning/afternoon/evening

  using the condition of the current time.

LOOPS

- Learn by yourself
  - https://www.tutorialspoint.com/csharp/csharp_loops.htm

1	**break statement** Terminatesthe **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
2	**continue statement** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

BEST PRACTICE

- Avoid multiple-nested loops!

  (Reason: Learn Big O again if you forget)

  (not required in this class:

  Optimise your workflow

  Change your data model

  Use stored procedures to do heavy tasks)

GROUP CODING LAB:
SELECT THREE CHECKBOXES
CLICK OK BUTTON TO SHOW A MESSAGE BOX WITH THE RESULT TEXT
HOW MANY LINES OF CODES FOR THE BUTTON?

Invoice	Statement	ticket		Result
Yes	No	Yes		Invoice
Yes	No	No		Invoice
Yes	Yes	Yes		Invoice and Statement
Yes	Yes	No		Invoice and Statement
No	Yes	Yes		Statement and Ticket
No	Yes	No		Statement
No	No	Yes		Ticket

ERROR HANDLING

- Next session

ASSIGNMENT

- Run and learn the codes I am sharing
- Will do an in-class test next time

Do not need to submit.