```
private void SimpleToPrintStringTests()
{
    // Simple tests case - no function arguments
    string output = "Test1".ToPrintString();
    Debug.WriteLine(
        string.Format("Characters in Test1 = {0}", output));

    // Declaring a simple array
    int[] SimpleArray = new int[] { 1, 2, 3, 4 };
    // Test Against an array with no arguments.
    output = SimpleArray.ToPrintString();
    Debug.WriteLine(
        string.Format("Simple 4 Element int array {0}", output));
    // Supplying a where clause
    output = SimpleArray.ToPrintString((val, pos) => val % 2 == 0);
    Debug.WriteLine(
        string.Format("Simple 4 Element int array, with a where clause {0}", output)); ;
    // Supplying an optional argument for the Format Function
    output = SimpleArray.ToPrintString(FormatFunction: (val, pos) => val.ToString());
    Debug.WriteLine(
        string.Format(
        "Simple 4 Element int array, with an optional Format Function argument\n{0}"
        , output)); ;
    // Supplying a optional argument for the Concatenation Function
    output = SimpleArray.ToPrintString(
        ConcatenateFunction: (carry, val) => carry.AppendFormat("{0}, ", val));
    Debug.WriteLine(
        string.Format(
        "Simple 4 Element int array, with an optional Concatenation Function argument\n{0}"
        , output));
    // Declaring a where predicate
    Func<int, int, bool> WherePredicate =
        (InputObject, Position) =>
        {
            if (Position % 2 == 0)
                return false;
            return true;
        };
```

```
// Supplying a where predicate as a externally declared function
output = SimpleArray.ToPrintString(WherePredicate);
Debug.WriteLine(
    string.Format(
    "Simple 4 Element int array, with a Where Predicate argument declared externally\n{0}"
    , output));

// Declaring a list of objects
List<Tuple<int, string>> SimpleObjects = new List<Tuple<int, string>>()
{
    Tuple.Create(1, "Test"),        Tuple.Create(200, "String Test"),
    Tuple.Create(-21, "Testing"),   Tuple.Create(0, "the quick brown")
};
// External Where Predicate
Func<Tuple<int, string>, int, bool> Where1 =
    (obj, pos) =>
    {
        if (obj.Item1 >= 0) return true;
        else return false;
    };
// External Format Predicate
Func<Tuple<int, string>, int, string> Format1 =
    (obj, pos) => string.Format(
        "[{0}] int value={1} string value ={2}\n"
        , pos, obj.Item1, obj.Item2);
// Calling using external declared predicates
output = SimpleObjects.ToPrintString(Where1, Format1);
Debug.WriteLine(
    string.Format("Processing a list of object with external predicates\n{0}"
    , output));
// Another where predicate
Func<Tuple<int, string>, int, bool> Where2 =
    (obj, pos) =>
    {
        if (obj.Item1 < 0) return true;
        else return false;
    };
// Calling using external declared predicates
```

```
    output = SimpleObjects.ToPrintString(Where2, Format1);
    Debug.WriteLine(
        string.Format("Processing a list of object with external predicates 2\n{0}"
        , output));

    // A more complex object collection to test against
    Dictionary<long, int?> DictTest = new Dictionary<long, int?>()
    {
        { 234L, null },         {-44345L, 65742 },
        { -5644, null },        {6799032L, 8765464 }
    };
    // Supplying where and format as more complex inline lambda expressions
    output = DictTest.ToPrintString((dictObj, pos) => dictObj.Value.HasValue,
        (dictObj, pos) =>
            string.Format("Key={0} Value={1}\n", dictObj.Key, dictObj.Value.Value));
    Debug.WriteLine(
        string.Format("Processing a dictionary with inline lambda expressions\n{0}"
        , output));
    // A format function for a dictionary.
    // NB: You need to unwrap the Dictionary into passed KeyValuePair objects.
    //      Also, lambda capture of the Dictionary Object to supply the Count property.
    Func<KeyValuePair<long, int?>, int, string> Format2 =
        (dictObj, position) =>
        {
            if (dictObj.Value.HasValue)
                return string.Format("Object {0} of {1} Key = {2} Value = {3}\n",
                    position, DictTest.Count(), dictObj.Key, dictObj.Value);
            else
                return string.Format("Object {0} of {1} Key = {2} Value = null\n",
                    position, DictTest.Count(), dictObj.Key);
        };
    output = DictTest.ToPrintString(FormatFunction: Format2);
    Debug.WriteLine(
        string.Format(
        "Processing the KeyValPair objects with named external function\n{0}"
        , output));
    return;
}
```