1. Describe the problem generics address.

   Generics address the problem of type safety and code reuse. They allow you to define classes, methods, and data structures without specifying the exact data types they will operate on. This leads to fewer runtime errors and eliminates the need for type casting, as the compiler can ensure type safety at compile time.

2. How would you create a list of strings, using the generic List class?

   List<string> stringList = new List<string>();

3. How many generic type parameters does the Dictionary class have?

   The Dictionary class has two generic type parameters: one for the key and one for the value.

   Dictionary<TKey, TValue>

4. True/False. When a generic class has multiple type parameters, they must all match.

   False. Each type parameter can be of a different type.

5. What method is used to add items to a List object?

   The Add method is used to add items to a List object.

   ```
   List<string> stringList = new List<string>();
   stringList.Add("Hello");
   ```

6. Name two methods that cause items to be removed from a List.

   - Remove: Removes the first occurrence of a specific object from the List.
   - RemoveAt: Removes the element at the specified index of the List.

7. How do you indicate that a class has a generic type parameter?

   You indicate that a class has a generic type parameter by using angle brackets (<>) after the class name and specifying a type parameter inside the brackets.

   ```
   public class GenericClass<T>
   {
           // Class implementation
   }
   ```

8. True/False. Generic classes can only have one generic type parameter.
   False. Generic classes can have multiple generic type parameters.

   ```
   public class GenericClass<T1, T2>
   {
           // Class implementation
   }
   ```

9. True/False. Generic type constraints limit what can be used for the generic type.

   True. Generic type constraints specify the requirements for the types that can be used as arguments for a generic type parameter.

   ```
   public class GenericClass<T> where T : IComparable
   {
           // Class implementation
   }
   ```

10. True/False. Constraints let you use the methods of the thing you are constraining to.

    True. Constraints allow you to use the methods and properties of the type specified in the constraint.

    ```
    public class GenericClass<T> where T : IComparable
    {
            public int Compare(T x, T y)
            {
                    return x.CompareTo(y);
            }
    }
    ```