This cheat sheet is for the course Learn C# Full Stack Development with Angular and ASP.NET by Jannick Leismann.

ASP.NET DATA ANNOTATION VALIDATION

You can apply validation rules directly to your model attributes in ASP.NET by using **data annotations**. In this manner, prior to processing or storing user input, you may ensure that it is accurate.

[Required]

This characteristic verifies the value of a property. A warning indicating an error will appear if the field is left empty.

```
public class User
{
    [Required(ErrorMessage = "Name is required")]
    public string Name { get; set; }
}
```

[StringLength]

This describes a string property's minimum and/or maximum length. It guarantees that the input is just the right length.

```
public class User
{
    [StringLength(50, MinimumLength = 5, ErrorMessage = "Name must be
between 5 and 50 characters")]
    public string Name { get; set; }
}
```

[Range]

This establishes the range of values that a numeric property may have. It can be used, for instance, to make sure a product's pricing falls inside a given range.

```
public class Product
{
    [Range(1, 100, ErrorMessage = "Price must be between 1 and 100")]
    public decimal Price { get; set; }
}
```

[RegularExpression]

By doing this, a property is checked against a given regular expression pattern. It is helpful for fields like usernames that must adhere to a specific format.

```
public class User
{
    [RegularExpression(@"^[a-zA-Z]+$", ErrorMessage = "Only letters are
allowed")]
    public string Name { get; set; }
}
```

[EmailAddress]

This ensures that a property contains a valid email address format.

```
public class User
{
    [EmailAddress(ErrorMessage = "Invalid email address")]
    public string Email { get; set; }
}
```

[Compare]

This attribute is used to compare the values of two properties. It’s typically used for password confirmation fields.

```
public class User
{
    [Required]
    public string Password { get; set; }
    [Compare("Password", ErrorMessage = "Passwords do not match")]
    public string ConfirmPassword { get; set; }
}
```

[CreditCard]

This validates that a property has a valid credit card number format.

```
public class Payment

{

    [CreditCard(ErrorMessage = "Invalid credit card number")]

    public string CreditCardNumber { get; set; }

}
```

[Phone]

This ensures that a property contains a valid phone number format.

```
public class User
{
    [Phone(ErrorMessage = "Invalid phone number")]
    public string PhoneNumber { get; set; }
}
```

[Url]

This ensures that a property contains a valid URL format.

```
public class Website
{
    [Url(ErrorMessage = "Invalid URL")]
    public string SiteUrl { get; set; }
}
```

Custom Validation

By deriving from the ValidationAttribute class, you can construct your own custom validation rule if necessary.

```
public class User
{
    [CustomValidation(typeof(UserValidator), "ValidateAge")]
    public int Age { get; set; }
}
```

```
public class UserValidator
{
    public static ValidationResult ValidateAge(int age, ValidationContext
context)
    {
        if (age < 18)
        {
            return new ValidationResult("Age must be 18 or older");
        }
        return ValidationResult.Success;
    }
}
```

These annotations strengthen and improve the readability of your code by assisting in ensuring that your data is accurate and satisfies the needs of your application.