1) What is a string, int and float in python?

In Python, data types are used to classify one particular type of data, determining the values that you can assign to the type and the operations you can perform on it. When programming, there are times we need to convert values between types in order to manipulate values in a different way. For example, we may need to concatenate numeric values with strings, or represent decimal places in numbers that were initialised as integer values.

1.1) String

A string in Python is a sequence of characters. It is a derived data type. Strings are immutable. This means that once defined, they cannot be changed. Many Python methods, such as replace(), join(), or split() modify strings. However, they do not modify the original string. They create a copy of a string which they modify and return to the caller.

Strings in python are surrounded by either single quotation marks, or double quotation marks.

1.2) Int

In Python, integers are zero, positive or negative whole numbers without a fractional part and having unlimited precision, e.g. 0, 100, -10.

Integers must be without a fractional part (decimal point). It includes a fractional then it becomes a float.

The int() function converts a string or float to int.

1.3) Float

In Python, floating point numbers (float) are positive and negative real numbers with a fractional part denoted by the decimal symbol, e.g. 1234.56, 3.142, -1.55, 0.23.

The float() function converts string, int to float.

2) What does a break do?

The break statement in Python terminates the current loop and resumes execution at the next statement, just like the traditional break found in C.

The break statement can be used in both *while* and *for* loops.

If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

3) What's the first index value in a list?

The items inside the list are indexed with the first element starting at index 0. You can make changes in the created list by adding new items or by updating, deleting the existing ones. It can also have duplicate items and a nested list.

4) What's the difference between a tuple and list?

In Python, list and tuple are a class of data structure that can store one or more objects or values. A list is used to store multiple items in one variable and can be created using square brackets. Similarly, tuples also can store multiple items in a single variable and can be declared using parentheses.

Although there are many differences between list and tuple, there are some similarities too, as follows:

4.1) List

List is just like the arrays, declared in other languages. Lists need not be homogeneous always which makes it the most powerful tool in Python. In Python, the list is a type of container in Data Structures, which is used to store multiple data at the same time. Lists are a useful tool for preserving a sequence of data and further iterating over it.

Eg: list_data = ['an', 'example', 'of', 'a', 'list']

4.2) Tuple

Tuple is also a sequence data type that can contain elements of different data types, but these are immutable in nature. In other words, a tuple is a collection of Python objects separated by commas. The tuple is faster than the list because of static in nature.

Eg: tuple_data = ('this', 'is', 'an', 'example', 'of', ‘tuple’)

5) Give us an example of CRUD operations?

The abbreviation CRUD expands to Create, Read, Update and Delete. These four are fundamental operations in a database.

We have created a student table having three fields: name, age and marks. The string holding the INSERT query is defined as:

insert_student_query = “INSERT INTO student (name, age, marks) VALUES (‘Bill’, 25, 50);”

The query string should contain the DELETE query syntax. For example, the below code is used to delete 'Bill' from the student table.

delete_student_query = “DELETE FROM student WHERE name=‘Bill’;”

6) What's the concept of DRY in django?

DRY, which stands for “**D**on’t **R**epeat **Y**ourself,” is a principle of software development that aims at reducing the repetition of patterns and code duplication in favour of abstractions and avoiding redundancy.

7) What is a POST method and where is this used in django?

In computing, POST is a request method supported by HTTP used by the World Wide Web. By design, the POST request method requests that a web server accepts the data enclosed in the body of the request message, most likely for storing it. It is often used when uploading a file or when submitting a completed web form.

Most common HTML forms on the web operate using this request method. It usually transmits relatively small loads of data to a receiver. This method allows data to be sent as a package in a separate communication with the processing script. This means that data sent through the POST method will not be visible in the URL, as parameters are not sent along with the URI.

A string representing the HTTP method used in the request. This is guaranteed to be uppercase. For example:

```
if request.method == 'GET':
    do_something()
elif request.method == 'POST':
    do_something_else()
```

8) What and how do you use the clean method in forms view?

Form validation happens when the data is cleaned. If we want to customize this process, there are various places we can change, each one serving a different purpose. Three types of cleaning methods are run during form processing. These are normally executed when we call the is_valid() method on a form. There are other things that can trigger cleaning and validation (accessing the errors attribute or calling full_clean() directly), but normally they won't be needed.

In general, any cleaning method can raise ValidationError if there is a problem with the data it is processing, passing the relevant error message to the ValidationError constructor. If no ValidationError is raised, the method should return the cleaned (normalized) data as a Python object.

If we detect multiple errors during a cleaning method and wish to signal all of them to the form submitter, it is possible to pass a list of errors to the ValidationError constructor.

The three types of cleaning methods are:

- The clean() method on a Field subclass. This is responsible for cleaning the data in a way that is generic for that type of field. For example, a FloatField will turn the data into a Python float or raise a ValidationError. This method returns the clean data, which is then inserted into the cleaned_data dictionary of the form.

- The clean_<fieldname>() method in a form subclass – where <fieldname> is replaced with the name of the form field attribute. This method does any cleaning that is specific to that particular attribute, unrelated to the type of field that it is. This method is not passed any parameters. You will need to look up the value of the field in self.cleaned_data and remember that it will be a Python object at this point, not the original string submitted in the form (it will be in cleaned_data because the general field clean() method, above, has already cleaned the data once).
  For example, if you wanted to validate that the contents of

a CharField called serialnumber was unique, clean_serialnumber() would be the right place to do this.

- The Form subclass's clean() method. This method can perform any validation that requires access to multiple fields from the form at once. This is where you might put in things to check that if field A is supplied, field B must contain a valid e-mail address and the like. The data that this method returns is the final cleaned_data attribute for the form, so don't forget to return the full list of cleaned data if you override this method (by default, Form.clean() just returns self.cleaned_data).

9) How can we use Ajax, jQuery to make CRUD operations in django?

Ajax is a way of making web development more dynamic. Using Ajax we can load data from the server without reloading the web pages. AJAX stands for Asynchronous Javascript and XML.

1) we have to import the JQuery in the base.html

```
<script src='https://code.jquery.com/jquery-3.2.1.min.js'></script>
```

2) Create the model

```
class CrudUser(models.Model):
    name = models.CharField(max_length=30, blank=True)
    address = models.CharField(max_length=100, blank=True)
    age = models.IntegerField(blank=True, null=True)
```

3) Create the url

```
path('ajax/crud/create/',  views.CreateCrudUser.as_view(), name='crud_ajax_create'),
```

4) Create the view

```
class CreateCrudUser(View):
    def  get(self, request):
        name1 = request.GET.get('name', None)
        address1 = request.GET.get('address', None)
        age1 = request.GET.get('age', None)

        obj = CrudUser.objects.create(
            name = name1,
            address = address1,
            age = age1
        )

        user = {'id':obj.id,'name':obj.name,'address':obj.address,'age':obj.age}

        data = {
            'user': user
        }
        return JsonResponse(data)
```

5) Example of Ajax

```
// Create Django Ajax Call
$("form#addUser").submit(function() {
    var nameInput = $('input[name="name"]').val().trim();
    var addressInput = $('input[name="address"]').val().trim();
    var ageInput = $('input[name="age"]').val().trim();
    if (nameInput && addressInput && ageInput) {
        // Create Ajax Call
        $.ajax({
            url: '{% url "crud_ajax_create" %}',
            data: {
```

```
            'name': nameInput,
            'address': addressInput,
            'age': ageInput
        },
        dataType: 'json',
        success: function (data) {
            if (data.user) {
              appendToUsrTable(data.user);
            }
        }
      });
    } else {
      alert("All fields must have a valid value.");
  }
  $('form#addUser').trigger("reset");
  return false;
});
```