PyObject

- PyObject is an object structure that you use to define object types for Python.
- All Python objects share a small number of fields that are defined using the PyObject structure.
- All other object types are extensions of this type.
- PyObject tells the Python interpreter to treat a pointer to an object as an object.

> For instance, setting the return type of the below function as PyObject defines the common fields that are required by the Python interpreter in order to recognize this as a valid Python type.

````
1  ```c
2  static PyObject *method_fputs(PyObject *self, PyObject *args) {
3      char *str, *filename = NULL;
4      int bytes_copied = -1;
5      /* Snip */
6
7  ```
````

- In line 2, you declare the argument types you wish to receive from your Python code:

  > 1. `char *str` is the string you want to write to the file stream.
  > 2. `char *filename` is the name of the file to write to.

PyArg_ParseTuple()

- `PyArg_ParseTuple()` parses the arguments you'll receive from your Python program into local variables:
-
  ```c
  1  static PyObject *method_fputs(PyObject *self, PyObject *args) {
  2      char *str, *filename = NULL;
  3      int bytes_copied = -1;
  4
  5      /* Parse arguments */
  6      if(!PyArg_ParseTuple(args, "ss", &str, &filename)) {
  7          return NULL;
  8      }
  9
  ```
-
- > If you look at line 6, then you'll see that `PyArg_ParseTuple()` takes the following arguments:

> - `args` are of type `PyObject`.
> - *so hapa `args` ni tuple and its values should follow the type expected which in this case is "ss" that is representing two strings,, **str and filename***
> - `"ss"` is the format specifier that specifies the data type of the arguments to parse. (You can check out the official documentation for a complete reference.)
> - `&str` **and** `&filename` are pointers to local variables to which the parsed values will be assigned.