In Python, the `__new__` method plays a crucial role in object creation. Let's dive into the details:

1. **Purpose of `__new__`**:
   - When you create an instance of a class, Python first calls the `__new__()` method to **create the object** and then calls the `__init__()` method to **initialize the object's attributes**.
   - The `__new__()` method is a **static method** of the `object` class.
   - Its signature is: `object.__new__(class, *args, **kwargs)`.
   - The first argument is the **class** of the new object you want to create, followed by optional `*args` and `**kwargs`.
2. **Creating a New Object**:
   - The `__new__()` method should return a new instance of the class.
   - You can override `__new__` to perform custom actions before and after creating an instance.
   - To create an object of a class, you typically call `super().__new__(...)`.
3. **Example**:

```
class Person:
def __init__(self, name):
    self.name = name

person = Person('John')
```

   - When you create an instance using `person = Person('John')`, Python internally calls both `__new__()` and `__init__()` methods.
   - Equivalent method calls:
     - `person = object.__new__(Person, 'John')`
     - `person.__init__('John')`
   - The `__dict__` of the `person` object:
     - After `__new__()`: `{}` (empty)
     - After `__init__()`: `{'name': 'John'}`
4. **Method Sequence**:
   - When creating an object by calling the class:

```
class Person:
def __new__(cls, name):
    print(f'Creating a new {cls.__name__}object...')
    obj = object.__new__(cls)
return obj

def __init__(self, name):
```

```
    print(f'Initializing the person object...')
    self.name = name

person = Person('John')
```

- Output:

```
Creating a new Person object...
Initializing the person object...
```

Remember, __new__ is responsible for object creation, while __init__ initializes the object's attributes. The order of execution is __new__ first, followed by __init__.