03 async nuances

Simple | This works fine, no problem at all

```
import requests

url = "https://pokeapi.co/api/v2/pokemon/pikachu"

def make_request():
    pikachu_response = requests.get(url)
    return pikachu_response.json()


make_request() # this works fine, no problem at all
```

You will get the TypeError: object Response can't be used in 'await' expression

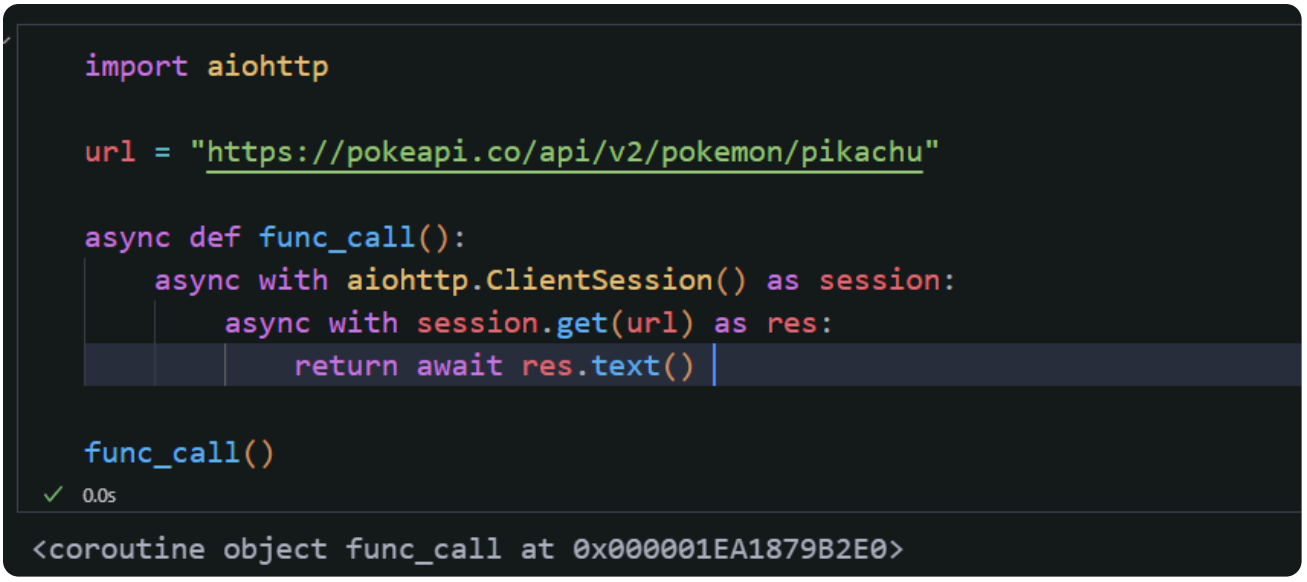
```
import asyncio
import requests

url = "https://pokeapi.co/api/v2/pokemon/pikachu"

async def make_request():
    pikachu_response = await requests.get(url)
    return pikachu_response.json()

make_request()


if __name__ == "__main__":
    result = asyncio.run(make_request())
    print(result)
```

Beasue:

The error occurs because we're trying to use the `await` keyword with a function (`requests.get`) that is **not asynchronous**.

- `requests.get(url)` is a **blocking, synchronous** function from the `requests` library.
- The `await` keyword only works with **coroutines** or **awaitable objects**, such as those returned by `async def` **functions** or by **asynchronous libraries** like `aiohttp`.
- Python raises an error because `requests.get(url)` returns a `requests.Response` object, which is **not awaitable**.

This does not work as it missing the event loop




```python
import aiohttp

url = "https://pokeapi.co/api/v2/pokemon/pikachu"

async def func_call():
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as res:
            return await res.text()

func_call()

# output: <coroutine object func_call at 0x000001EA1879B2E0>
```

This works, as it has the eventloop

When we call `func_call()` directly, it returns a coroutine object (as shown in the output `<coroutine object func_call at 0x000001EA1879B2E0>`), but it doesn't execute the function.

To fix this, we need to run the coroutine in an event loop.

```python
import aiohttp
import asyncio

url = "https://pokeapi.co/api/v2/pokemon/pikachu"

async def func_call():
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as res:
            return await res.text()

if __name__ == "__main__":
    result = asyncio.run(func_call())
    print(result)
```

The `asyncio.run()` function takes care of creating the event loop, running your coroutine until it's complete, and then closing the loop.