1. Dynamic SQL Query Builder

Problem Statement:

Create a function to dynamically build an SQL query. The function should take a table name, column names as positional arguments (`*args`), and conditions as keyword arguments (`**kwargs`). If no columns are provided, the query should default to selecting all columns (`*`).

-> def build_query(table, *columns, **conditions):

Sample Input:

```
build_query("users", "id", "name", "email", status="active", role="admin")
```

Sample Output:

```
SELECT id, name, email FROM users WHERE status='active' AND role='admin'
```

---

2. Dynamic API Request Simulator

Problem Statement:

Simulate an API request function that takes a URL as a required parameter, optional additional data as positional arguments (`*args`), and headers or query parameters as keyword arguments (`**kwargs`). Print the URL, additional data, and headers/parameters.

-> def api_request(url, *args, **kwargs):

Sample Input:

```
api_request("https://api.example.com/data", "extra_info", token="123ABC",
timeout=30)
```

Sample Output:

```
Requesting: https://api.example.com/data
Args (for optional data): ('extra_info',)
Headers/Params: {'token': '123ABC', 'timeout': 30}
```

3. Custom Event Logger

Problem Statement:

Write a function that logs events with varying levels of details. Use positional arguments (`*args`) for optional event details and keyword arguments (`**kwargs`) for metadata about the event.

-> def log_event(event_name, *details, **metadata):

Sample Input:

```
log_event("UserLogin", "Attempt by admin", ip="192.168.1.1", status="Success",
timestamp="2024-11-20")
```

Sample Output:

```
Event: UserLogin
Details: Attempt by admin
Metadata:
  ip: 192.168.1.1
  status: Success
  timestamp: 2024-11-20
```

4. Dynamically Configurable Web Page Generator

Problem Statement:

Design a function that generates a simple web page with a title, optional sections (`*args`), and optional styles (`**kwargs`). The sections will be enclosed in `<section>` tags, and styles should be converted to inline CSS.

-> def generate_webpage(title, *sections, **styles):

Sample Input:

```
generate_webpage(
    "My Website",
    "Welcome to my website!",
    "Here is some content.",
    color="blue", font_size="16px", background_color="lightgray"
)
```

Sample Output:

```
<html>
<head>
    <title>My Website</title>
    <style>color: blue; font_size: 16px; background_color: lightgray;</style>
</head>
<body>
    <section>Welcome to my website!</section>
    <section>Here is some content.</section>
</body>
</html>
```

---

5. Multi-Language Translator

Problem Statement:

Create a function to translate a phrase into multiple languages. Use `*args` to specify the target languages and `**kwargs` to provide translations for each language. If a translation is not available for a language, return a default message.

-> def translate(phrase, *languages, **translations):

Sample Input:

```
translate("greet", "en", "es", "de", en="Hello", es="Hola", fr="Bonjour")
```

Sample Output:

```
{'en': 'Hello', 'es': 'Hola', 'de': "[Translation for 'de' unavailable]"}
```