

FastAPI Examples



DR. MOHAMMED AL-HUBAISHI

Introduction to FastAPI Examples

These simple examples to help you understand the FastAPI framework better.

1. Hello World Example
2. Square of a Number
3. String Length
4. Sum of Two Numbers
5. Check Prime Number
6. Concatenate Strings
7. Check Palindrome
8. Generate a Fibonacci Sequence
9. Reverse a String
10. Convert Celsius to Fahrenheit

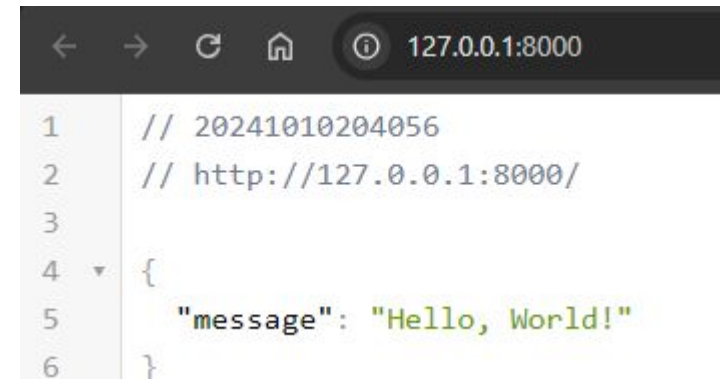
<https://github.com/DrMohammedhbi/FastApiExamples>

Hello World Example

```
> fastapi dev .\HelloWorld.py
```

```
from fastapi import FastAPI
app = FastAPI()

@app.get('/')
async def root():
    return {'message': 'Hello, World!'}
```



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000'. The page content is a JSON response from the root endpoint, showing a timestamp, the URL, and a message.

```
1 // 20241010204056
2 // http://127.0.0.1:8000/
3
4 {
5   "message": "Hello, World!"
6 }
```

- Explanation: The root endpoint ('/') returns a JSON message {'message': 'Hello, World!'}

Square of a Number

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get('/square/{num}')  
async def square(num: int):  
    return {'square': num ** 2}
```

- Explanation: The path parameter num is squared and returned in the response.

GET /square/{num} Square

Parameters

Name	Description
num * required	
integer (path)	5

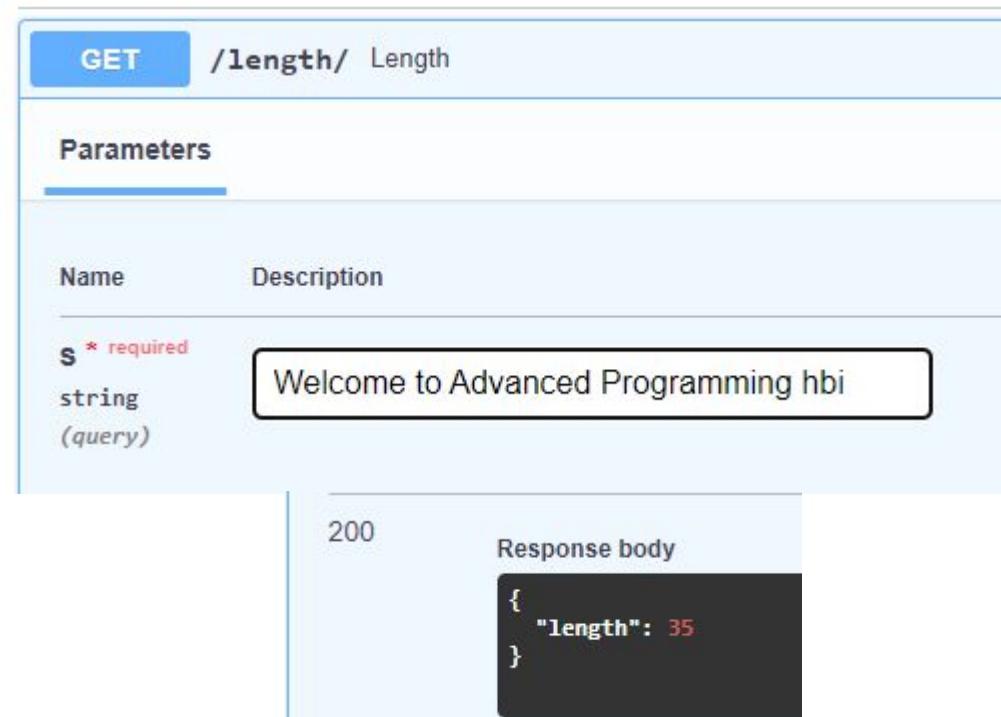
200

Response body

```
{  
  "square": 25  
}
```

```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/square/5  
{"square":25}  
C:\Users\hbi>
```

String Length



The image shows a Swagger UI interface for a REST API endpoint. At the top, it indicates a GET method for the path `/length/` with the title 'Length'. Below this, there is a 'Parameters' section. It contains a table with two columns: 'Name' and 'Description'. A single parameter is listed: 's', which is a required string (indicated by a red asterisk and the word 'required' in red). The parameter is of type 'string' and is a query parameter (indicated by '(query)'). To the right of the parameter table, there is a text input field containing the value 'Welcome to Advanced Programming hbi'. Below the parameters section, the response information is shown: a status code of 200 and a 'Response body' section. The response body is a JSON object: `{ "length": 35 }`.

Name	Description
s * required string (query)	Welcome to Advanced Programming hbi

200

Response body

```
{  
  "length": 35  
}
```

5

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get('/length/')
```

```
async def length(s: str):
```

```
    return {'length': len(s)}
```

```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/length/?s=Welcome%20to%20Advanced%20Programming%20hbi  
{ "length": 35 }  
C:\Users\hbi>
```

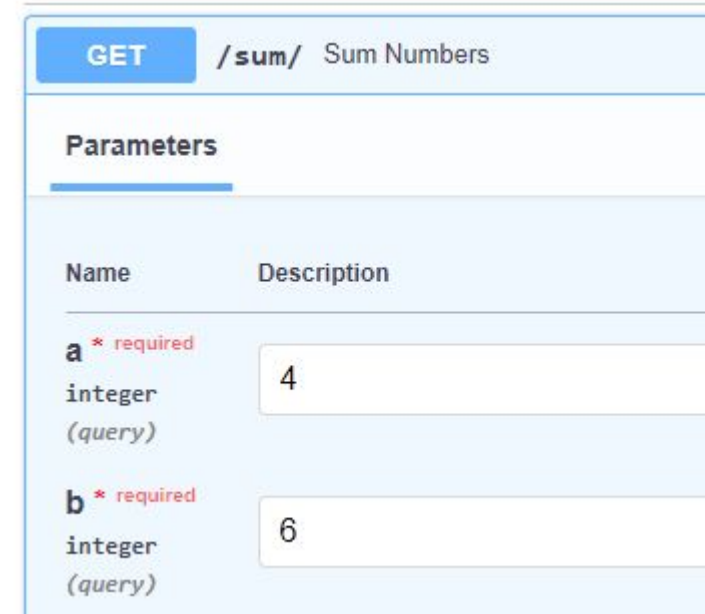
- Explanation: The query parameter `s` (a string) is measured, and the length is returned.

Sum of Two Numbers

```
from fastapi import FastAPI
app = FastAPI()
```

```
@app.get('/sum/')
async def sum_numbers(a: int, b: int):
    return {'sum': a + b}
```

- Explanation: Two integers, a and b, are passed as query parameters, and their sum is returned.



GET /sum/ Sum Numbers

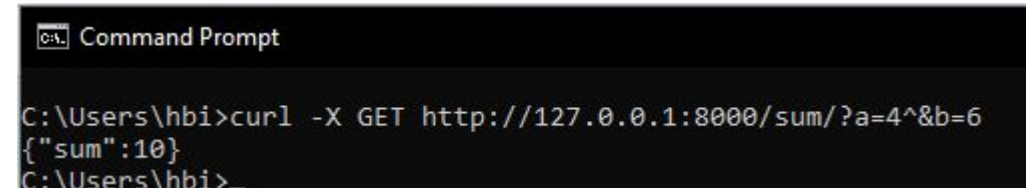
Parameters

Name	Description
a * required integer (query)	4
b * required integer (query)	6

6

Response body

```
{
  "sum": 10
}
```



```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/sum/?a=4&b=6
{"sum":10}
C:\Users\hbi>
```

Check Prime Number

```
1
2 from fastapi import FastAPI, HTTPException
3
4 app = FastAPI()
5
6 def is_prime(n: int) -> bool:
7     if n < 2:
8         return False
9     for i in range(2, int(n**0.5) + 1):
10         if n % i == 0:
11             return False
12     return True
13
14 @app.get("/prime/{num}")
15 async def prime(num: int):
16     if num < 1:
17         raise HTTPException(status_code=400, detail="Input must be greater than 0.")
18     return {"is_prime": is_prime(num)}
19
```

```
CA: Command Prompt
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/prime/9
{"is_prime":false}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/prime/5
{"is_prime":true}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/prime/6
{"is_prime":false}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/prime/2
{"is_prime":true}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/prime/3
{"is_prime":true}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/prime/7
{"is_prime":true}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/prime/8
{"is_prime":false}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/prime/22
{"is_prime":false}
C:\Users\hbi>
```

- Explanation: Checks if num is prime and returns the result. Validates that the number is greater than 0.

Concatenate Strings

8

Name	Description
a * required string (query)	we
b * required string (query)	are

Response body

```
{  
  "concatenated": "weare"  
}
```

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get('/concat/')
```

```
async def concat_strings(a: str, b: str):  
    return {'concatenated': a + b}
```

```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/concat/?a=we^&b=areIntheclass  
{"concatenated":"weareIntheclass"}  
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/concat/?a=How^&b=areYou  
{"concatenated":"HowareYou"}  
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/concat/?a=Advanced^&b=computer programming  
{"concatenated":"Advancedcomputer"}curl: (6) Could not resolve host: programming  
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/concat/?a=Advanced^&b=computerProgramming  
{"concatenated":"AdvancedcomputerProgramming"}  
C:\Users\hbi>
```

- Explanation: Takes two strings a and b as query parameters and returns their concatenation.

Check Palindrome

```
200 OK
> fastapi dev .\CheckPalindrome.py
```

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get('/palindrome/{word}')
```

```
async def check_palindrome(word: str):
```

```
    is_palindrome = word == word[::-1]
```

```
    return {'is_palindrome': is_palindrome}
```

```
1 // 20241010202313
2 // http://127.0.0.1:8000/palindrome/level
3
4 {
5     "is_palindrome": true
6 }
```

```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/palindrome/racecar
{"is_palindrome":true}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/palindrome/level
{"is_palindrome":true}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/palindrome/welcome
{"is_palindrome":false}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/palindrome/nerom
{"is_palindrome":false}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/palindrome/civic
{"is_palindrome":true}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/palindrome/noon
{"is_palindrome":true}
C:\Users\hbi>
```

► Explanation: The path parameter word is checked if it reads the same forwards and backwards.

► "racecar""level""deified""radar""madam""civic""kayak""rotator""noon""deed"

<http://localhost:8000/palindrome/racecar>
<http://localhost:8000/palindrome/level>
<http://localhost:8000/palindrome/deified>

Generate a Fibonacci Sequence

```
> fastapi dev .\FibonacciSequence.py
```

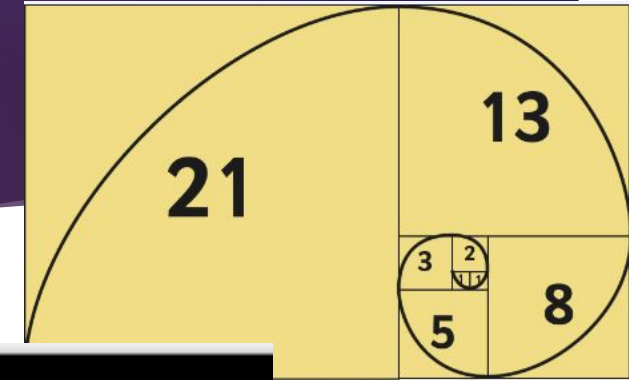
```
from fastapi import FastAPI
app = FastAPI()

def fibonacci(n: int):
    sequence = [0, 1]
    while len(sequence) < n:
        sequence.append(sequence[-1] + sequence[-2])
    return sequence[:n]

@app.get('/fibonacci/{count}')
async def generate_fibonacci(count: int):
    if count <= 0:
        return {'error': 'Count must be a positive integer.'}
    return {'fibonacci': fibonacci(count)}
```

Command Prompt

```
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/fibonacci/2
{"fibonacci": [0, 1]}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/fibonacci/3
{"fibonacci": [0, 1, 1]}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/fibonacci/4
{"fibonacci": [0, 1, 1, 2]}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/fibonacci/7
{"fibonacci": [0, 1, 1, 2, 3, 5, 8]}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/fibonacci/9
{"fibonacci": [0, 1, 1, 2, 3, 5, 8, 13, 21]}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/fibonacci/15
{"fibonacci": [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]}
C:\Users\hbi>
```



- Explanation: The path parameter count is used to generate and return a Fibonacci sequence of that length. For example, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377.

Reverse a String

```
> fastapi dev .\ReverseString.py
```

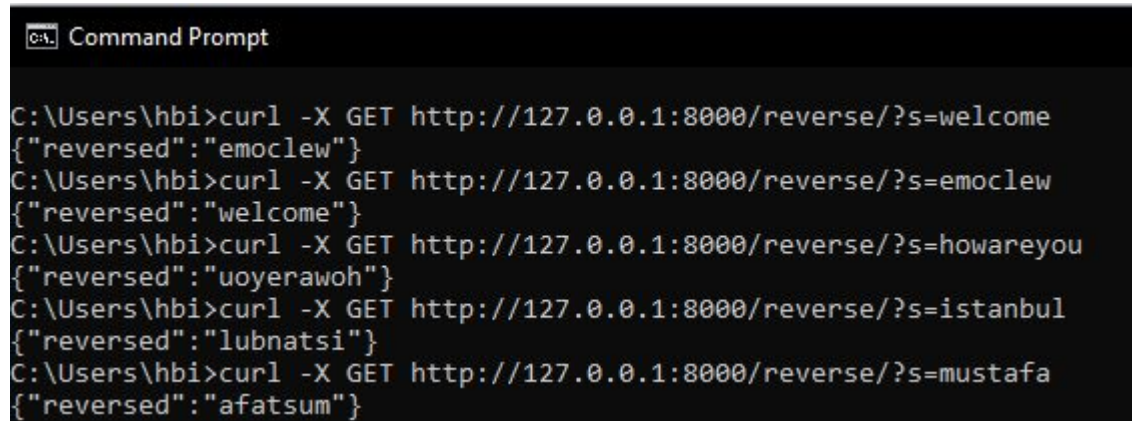
```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get('/reverse/')
```

```
async def reverse_string(s: str):  
    return {'reversed': s[::-1]}
```

- Explanation: The string `s` passed as a query parameter is reversed and returned.



```
Command Prompt  
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/reverse/?s=welcome  
{  
  "reversed": "emoclew"  
}  
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/reverse/?s=emoclew  
{  
  "reversed": "welcome"  
}  
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/reverse/?s=howareyou  
{  
  "reversed": "uoyerawoh"  
}  
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/reverse/?s=istanbul  
{  
  "reversed": "lubnatsi"  
}  
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/reverse/?s=mustafa  
{  
  "reversed": "afatsum"  
}
```

Convert Celsius to Fahrenheit

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get('/convert/')
```

```
async def convert_celsius_to_fahrenheit(celsius: float):
```

```
    fahrenheit = (celsius * 9/5) + 32
```

```
    return {'fahrenheit': fahrenheit}
```



```
Command Prompt

C:\Users\hbi>curl -X GET http://127.0.0.1:8000/convert/?celsius=23
{"fahrenheit":73.4}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/convert/?celsius=2
{"fahrenheit":35.6}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/convert/?celsius=3
{"fahrenheit":37.4}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/convert/?celsius=6
{"fahrenheit":42.8}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/convert/?celsius=40
{"fahrenheit":104.0}
C:\Users\hbi>curl -X GET http://127.0.0.1:8000/convert/?celsius=18
{"fahrenheit":64.4}
C:\Users\hbi>
```

- Explanation: Takes a celsius value as a query parameter and returns the corresponding temperature in Fahrenheit.

Euclid's Algorithm - gcd

- ▶ Euclid's algorithm is based on the fact that if u is greater than v then the greatest common divisor of u and v is the same as the **greatest common divisor** of v and $u \% v$.
- ▶ This description explains how to compute the greatest common divisor of two numbers by computing the greatest common divisor of two smaller numbers.
- ▶ We can implement this method in FastApi python simply by having the **gcd function**
- ▶ **gcd(461952,116298);**

```
from fastapi import FastAPI, HTTPException
app = FastAPI()

# Euclidean algorithm to calculate GCD
def euclidean_algorithm(u: int, v: int) -> int:
    while v != 0:
        u, v = v, u % v
    return u

@app.get("/gcd/")
async def calculate_gcd(a: int, b: int):
    if a <= 0 or b <= 0:
        raise HTTPException(status_code=400, detail="Both numbers

gcd_value = euclidean_algorithm(a, b)
return {"GCD": gcd_value}
```

(461952, 116298)
(116298, 113058)
(113058, 3240)
(3240, 2898)
(2898, 342)
(342, 162)
(162, 18)
(18, 0)

```

from fastapi import FastAPI, HTTPException
app = FastAPI()

# Euclidean algorithm to calculate GCD
def euclidean_algorithm(u: int, v: int) -> int:
    while v != 0:
        u, v = v, u % v
    return u

@app.get("/gcd/")
async def calculate_gcd(a: int, b: int):
    if a <= 0 or b <= 0:
        raise HTTPException(status_code=400, detail="Both numbers

gcd_value = euclidean_algorithm(a, b)
return {"GCD": gcd_value}

```

curl -X GET http://127.0.0.1:8000/gcd/?a=461952^&b=116298

Command Prompt

```

C:\Users\hbi>curl -X GET http://127.0.0.1:8000/gcd/?a=461952^&b=116298
{"GCD":18}
C:\Users\hbi>

```

GET /gcd/ Calculate Gcd

Parameters

Name	Description
a * required integer (query)	<input type="text" value="461952"/>
b * required integer (query)	<input type="text" value="116298"/>

Dr.Mohammed AlHubaishi

Request URL

http://127.0.0.1:8000/gcd/?a=461952&b=116298

Server response

Code

Details

200

Response body

```

{
  "GCD": 18
}

```

Response headers

References

<https://fastapi.tiangolo.com/>

<https://github.com/DrMohammedhbi/FastApiExamples>