LẬP TRÌNH PYTHON NÂNG CAO

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm biên soạn:

TP.HCM - Năm 2021

MỤC LỤC

[BÀI 8: LẬP TRÌNH DỮ LIỆU MẠNG INTERNET 3](#_Toc80708833)

[1. Dữ liệu Internet và mô hình Request/Response 3](#_Toc80708834)

[1.1. HTTP 3](#_Toc80708835)

[1.2. HTTP Message 4](#_Toc80708836)

[1.2.1. HTTP Request 4](#_Toc80708837)

[1.2.2. HTTP Response 5](#_Toc80708838)

[2. Thư viện urllib trong python 6](#_Toc80708839)

[2.1. urllib request 6](#_Toc80708840)

[2.1.1. urlopen 6](#_Toc80708841)

[2.1.2. urllib.parse 7](#_Toc80708842)

[2.2. Xử lý ngoại lệ 8](#_Toc80708843)

[3. Rest API và Webservice 9](#_Toc80708844)

[3.1. Kiến trúc của REST 9](#_Toc80708845)

[3.2. REST APIs and Web Services 10](#_Toc80708846)

[3.3. REST và Python 10](#_Toc80708847)

[3.3.1. Phương thức get 10](#_Toc80708848)

[3.3.2. Phương thức post 11](#_Toc80708849)

[3.3.3. Phương thức put 13](#_Toc80708850)

[3.3.4. Phương thức Patch 14](#_Toc80708851)

[3.3.5. Phương thức delete 14](#_Toc80708852)

[3.4. JSON và XML 14](#_Toc80708853)

[BÀI TẬP CHƯƠNG 8 16](#_Toc80708854)

# BÀI 8: LẬP TRÌNH DỮ LIỆU MẠNG INTERNET

**Mục tiêu:**

* *Nắm vững được Python để viết các đoạn lệnh.*
* *Hiểu được khái niệm API và định dạng dữ liệu JSON/XML.*
* *Sử dụng được gói request, urllib để thực hiện các tác vụ truy vấn dữ liệu Internet.*

**Nội dung chính:**

# Dữ liệu Internet và mô hình Request/Response

## HTTP

HTTP (Hyper-Text-Transfer-Protocol) là một giao thức mức ứng dụng cho các hệ thống phân phối, cộng tác, đa phương tiện. Nó được thiết kế bởi Tim Berners Lee (1989). Giao thức này dựa trên giao thức TCP/IP. Đây là nền tảng đã được World-Wide Web (WWW) sử dụng từ năm 1990, cho phép tải (fetching) các nguồn tài nguyên trên mạng như tài liệu HTML, text, video, ảnh.

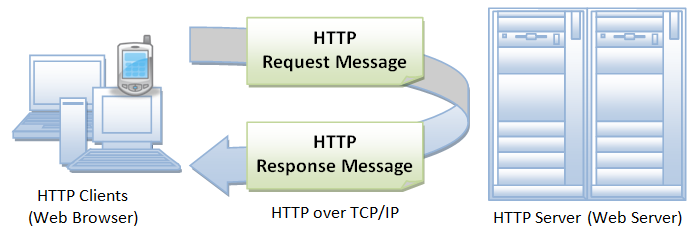
Nó cũng là nền tảng cho bất kỳ trao đổi dữ liệu nào được thực hiện trên Web với mô hình Client/Server. Các yêu cầu (Request/HTTP Request) từ client được gửi đến Server, sau đó Server xử lý yêu cầu và gửi lại (Response/HTTP Response) dữ liệu đến client. Các Request và Response là các Message (HTTP Message).

**Stateless**: HTTP là giao thức phi trạng thái (stateless), không có mối liên hệ nào giữa hai yêu cầu (request) được gửi đi, dù nó thực hiện trên cùng một kết nối (đến cùng 1 server). Thí dụ, client, thông qua web browser (như chrome/firefore) gửi yêu cầu đến <https://www.vanlanguni.edu.vn/dao-tao/nganh-dao-tao>, tiếp tục nó có thể gửi một yêu cầu khác đến <https://www.vanlanguni.edu.vn/trang-chu/lich-su>, cả hai yêu cầu này không có mối liên hệ gì với nhau.

**Cookie**: Đôi khi một ứng dụng cần chia sẻ trạng thái giữa những yêu cầu của phiên làm việc (session). Chẳng hạn trong ứng dụng bán hàng online, người dùng thêm 1 sản phẩm vào giỏ. Sau đó, quay lại trang chính để tìm kiếm và thêm sản phẩm khác vào giỏ. Do HTTP là stateless, nên HTTP Cookie sẽ hỗ trợ nhằm duy trì trạng thái của hai phiên làm việc trên. Phần Cookie này được thêm vào phần header của HTTP Message.

## HTTP Message

Thông điệp HTTP (HTTP Message) là thông tin được trao đổi qua lại giữa Client và Server. Trong đó, HTTP Request Message là thông điệp gửi từ Client đến Server, và ngược lại là HTTP Response Message.

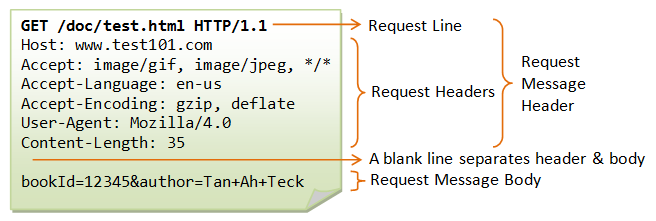


Phiên bản HTTP/1.1 của HTTP Message được chuẩn hóa vào năm 1997, đây là phiên bản giao thức HTTP mặc định trong các trình duyệt. Trong khi đó, HTTP/2.0 chuẩn hóa năm 2015 là phiên bản cải tiến tiếp theo. Nếu Server hỗ trợ HTTP/2.0 thì các trình duyệt sẽ tự động sử dụng phiên bản HTTP/2.0.

## HTTP Request

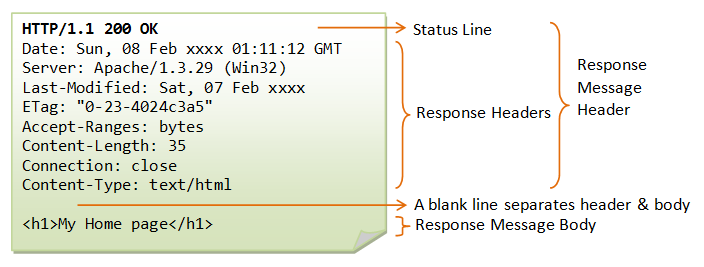
HTTP Request là thông báo yêu cầu từ máy khách gữi đến máy chủ. Nội dung thông báo bao gồm: Request Line, Request Header và Request Body (tùy chọn).

* **Request Line**:
  + Tên phương thức: GET, POST, PUT, DELETE
  + URL: là địa chỉ định danh tài nguyên
  + HTTP version: phiên bản của giao thức HTTP (ví dụ HTTP/1.0, HTTP/1.1)
* **Request Header**: cho phép client gửi thêm các thông tin bổ sung đính kèm với HTTP Request, Một số trường thông dụng như:
  + Accept loại nội dung có thể nhận được từ thông điệp response (ví dụ: text/plain, text/html…)
  + Accept-Encoding: các kiểu nén được chấp nhận (ví dụ gzip, deflate)
  + Connection: tùy chọn điều khiển cho kết nối hiện thời. Ví dụ: keepalive, Upgrade…
  + Cookie: thông tin HTTP Cookie từ server
  + User-Agent: thông tin về user agent của người dùng
* **Request Body** (tùy chọn): Các yêu cầu tìm nạp tài nguyên, như GET, HEAD, DELETE hoặc OPTIONS, thường không cần Request Body.



## HTTP Response

HTTP Response cũng là một HTTP Message, nên nó cũng có cấu trúc tương tự với HTTP Resquest, gồm các thành phần Response Line, Response Header và Response Body.



Response Line có cấu trúc như sau:

HTTP-version – Status Code – Reason Phrase

|  |  |
| --- | --- |
| HTTP-version | Phiên bản HTTP, thông dụng nhất là HTTP/1.1 |
| Status Code | Mã phản hồi do máy chủ gửi về. Ví dụ: 200, 301, 500 |
| Reason Phrase | Giải thích ngắn gọn về mã phản hồi |

# Thư viện urllib trong python

Module urllib trong Python 3 cho phép người lập trình để truy cập các trang web. Thông qua urllib, người phát triển ứng dụng có thể truy cập các trang web, tải xuống dữ liệu, phân tích cú pháp dữ liệu, sửa đổi tiêu đề và thực hiện bất kỳ yêu cầu GET và POST nào mà họ có thể cần thực hiện. Dưới đây là bảng liệt kê một số package của module urllib.

|  |  |
| --- | --- |
| **Modules** | **Mô tả** |
| urllib.request | mở và đọc |
| urllib.parse | phân tích cú pháp URL |
| urllib.error | cho các trường hợp ngoại lệ được nêu ra |
| urllib.robotparser | phân tích cú pháp tệp robot.txt |

(Các demo trong những thí dụ dưới đây sinh viên có thể tham khảo tại link: <https://github.com/huynhhoc/AdvancedPythonProgramming/tree/main/Chapter%208>)

Cài đặt urllib:

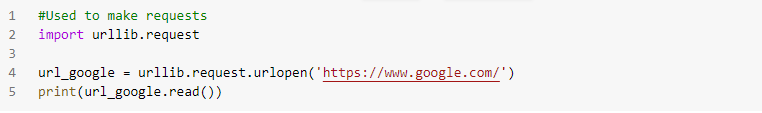
pip install urllib

# urllib request

# urlopen

urlopen dùng để mở một đối tượng mạng thông qua chuỗi URL. Trường hợp kết nối thành công, một đối tương giống như tập tin sẽ được trả về, nếu kết nối không thể được thực hiện, ngoại lệ IOError được đưa ra. Đôi tượng trả về hỗ trợ các phương thức sau: read (), readline (), readlines (), fileno (), close (), info (), getcode () và geturl ().

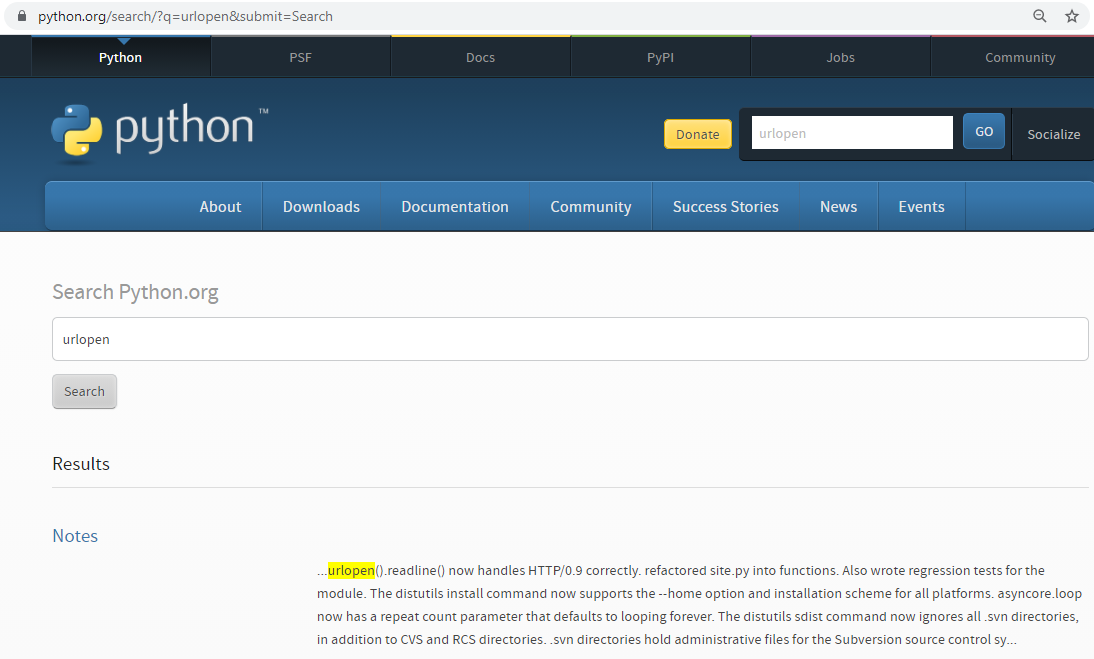
**Thí dụ 1**:



**Thí dụ 2**: Sinh viên hãy phát triển trên thí dụ 1 bằng cách sử dụng các hàm info(), getcode() và geturl(), sau đó ghi lại kết quả và giải thích ý nghĩa của từng hàm.

# urllib.parse

Giả sử chúng ta muốn tìm kiếm thông tin ‘basic’ từ trang web [www.python.org](http://www.python.org), chúng ta có thể vào trang đó và gõ từ **openurl** vào trong mục tìm kiếm, rồi nhấn vào nút **Seach,** khi đó địa chỉ trang có dạng là <https://www.python.org/search/?q=urlopen&submit=Search> (xem hình minh họa dưới đây).



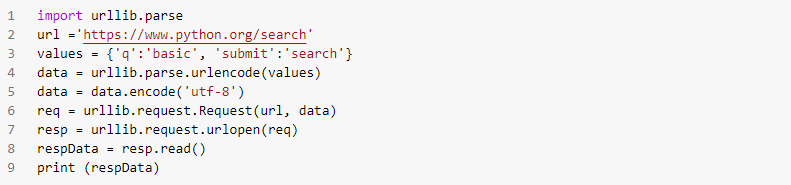
Khi đó để lấy được nội dung của kết quả tìm kiếm, ta xem thí dụ 2 dưới đây:

**Thí dụ 2**:



Bên cạnh đó, chúng ta có thể sử dụng thư viện **urllib.parse** để thay thế, hãy xem thí dụ 3 dưới đây (kết quả thí dụ 3 hoàn toàn tương tự thí dụ 2)

**Thí dụ 3**:



# Xử lý ngoại lệ

urlopen có thể gây ra ngoại lệ (exception) URLError trong trường hợp máy chủ tại địa chỉ url không phản hồi kết quả. Khi đó, người phát triển ứng dụng nên nắm bắt và xử lý các ngoại lệ này. Xem thí dụ 4 và 5 dưới đây.

**Thí dụ 4**:



**Thí dụ 5**:



# Rest API và Webservice

# Kiến trúc của REST

REST (representational state tranfer) là một dạng kiến trúc phần mềm xác định kiểu cho giao tiếp giữa client và server thông qua mạng. REST cung cấp một tập các ràng buộc cho kiến trúc phần mềm để thúc đẩy hiệu suất, khả năng mở rộng, tính đơn giản và độ tin cậy trong hệ thống.

REST định nghĩa các ràng buộc sau:

* **Stateless**: Máy chủ không duy trì bất cứ trạng thái nào giữa các yêu cầu từ máy khách.
* **Client-Server**: Máy chủ và máy khách phải phát triển độc lập với nhau.
* **Cacheable**: Dữ liệu được truy xuất từ máy chủ phải được máy khách hoặc máy chủ lưu vào bộ nhớ cache.
* **Uniform interface**: Máy chủ sẽ cung cấp một giao diện thống nhất để truy cập tài nguyên mà không cần xác định đại diện của chúng.
* **Layered system**: Máy khách có thể truy cập tài nguyên trên máy chủ một cách gián tiếp thông qua các tầng khác nhau như proxy hoặc load balancer[[1]](#endnote-1).
* **Code on demand (tùy chọn)**: Máy chủ có thể chuyển mã nguồn đến máy khách mà nó có thể chạy, chẳng hạn như JavaScript cho ứng dụng một trang.

# REST APIs and Web Services

Web service REST là bất kỳ dịch vụ web nào tuân theo các ràng buộc về kiến trúc REST. Các webservice này hiển thị dữ liệu của họ với thế giới bên ngoài thông qua một API. API REST cung cấp quyền truy cập vào dữ liệu dịch vụ web thông qua các URL web công khai.

Các API REST lắng nghe các phương thức HTTP như GET, POST và DELETE để biết các thao tác nào cần thực hiện trên tài nguyên của web service. Tài nguyên là bất kỳ dữ liệu nào có sẵn trong webservice có thể được truy cập và thao tác với các yêu cầu HTTP tới API REST. Phương thức HTTP cho API biết hành động nào cần thực hiện trên tài nguyên.

| **HTTP method** | **Description** |
| --- | --- |
| GET | Retrieve an existing resource. |
| POST | Create a new resource. |
| PUT | Update an existing resource. |
| PATCH | Partially update an existing resource. |
| DELETE | Delete a resource. |

**API Endpoints** API REST hiển thị một tập hợp các URL công khai mà các ứng dụng khách sử dụng để truy cập các tài nguyên của web service. Các URL này, trong ngữ cảnh của một API, được gọi là Endpoint

[**https://docs.python-requests.org/en/master/user/quickstart/#make-a-request**](https://docs.python-requests.org/en/master/user/quickstart/#make-a-request)

# REST và Python

Để lập trình tương tác với REST API, những nhà phát triển Python có thể sử dụng requests để gửi HTTP requests. Nó là một thư viện HTTP được cấp phép của Apache2, được viết bằng Python.

Cài đặt thư viện requests:

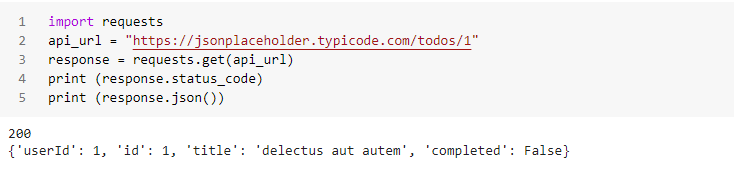
pip install requests

# Phương thức get

GET là một trong số các phương thức HTTP phổ biến mà người lập trình sử dụng khi làm việc với REST API. Phương thức này chỉ cho phép truy xuất tài nguyên từ một API cho trước, không cho phép thay đổi hoặc chỉnh sửa tài nguyên.

Người lập trình có thể sử dụng dịch vụ giả [JSONPlaceholder](https://jsonplaceholder.typicode.com/) (<https://jsonplaceholder.typicode.com>) để kiểm tra GET.

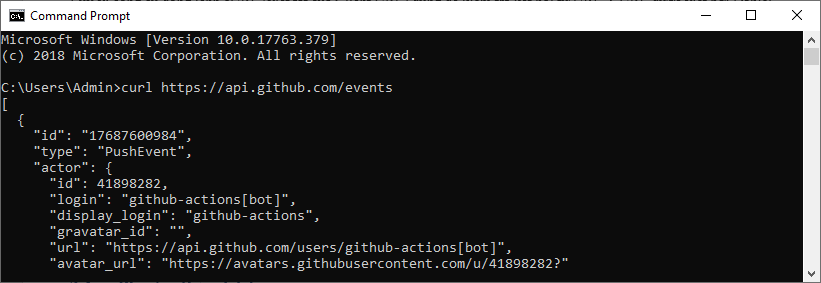
**Thí dụ 6**:



Đoạn mã này gọi request.get () để gửi yêu cầu GET tới /todos/1, yêu cầu này sẽ phản hồi với mục **todo** với **ID** 1. Trong thí dụ này, dữ liệu trả về có định dạng JSON.

**Câu hỏi**: làm sao để có thể xem headers của response trả về:

curl https://api.github.com/events



# Phương thức post

Now, take a look at how you use requests to POST data to a REST API to create a new resource. You’ll use JSONPlaceholder again, but this time you’ll include JSON data in the request. Here’s the data that you’ll send:



Here, you call requests.post() to create a new todo in the system.

First, you create a dictionary containing the data for your todo. Then you pass this dictionary to the json keyword argument of requests.post(). When you do this, requests.post() automatically sets the request’s HTTP header Content-Type to application/json. It also serializes todo into a JSON string, which it appends to the body of the request.

If you don’t use the json keyword argument to supply the JSON data, then you need to set Content-Type accordingly and serialize the JSON manually. Here’s an equivalent version to the previous code:



In this code, you add a headers dictionary that contains a single header Content-Type set to application/json. This tells the REST API that you’re sending JSON data with the request.

You then call requests.post(), but instead of passing todo to the json argument, you first call json.dumps(todo) to serialize it. After it’s serialized, you pass it to the data keyword argument. The data argument tells requests what data to include in the request. You also pass the headers dictionary to requests.post() to set the HTTP headers manually.

When you call requests.post() like this, it has the same effect as the previous code but gives you more control over the request.

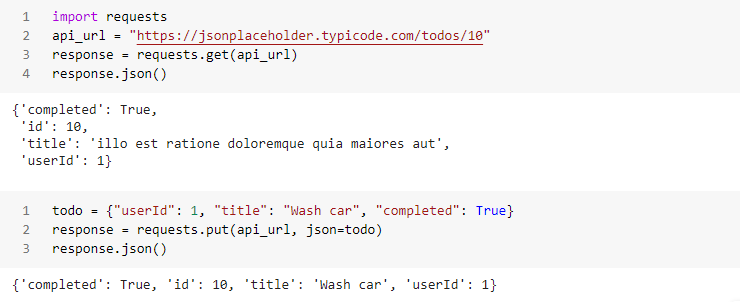
**Note:** [json.dumps()](https://docs.python.org/3/library/json.html#json.dumps) comes from the [json](https://docs.python.org/3/library/json.html) package in the standard library. This package provides useful methods for working with [JSON in Python](https://realpython.com/python-json/).

# Phương thức put

Bên cạnh GET và POST, requests còn hỗ trợ Phương thức PUT.

Beyond GET and POST, requests provides support for all the other HTTP methods you would use with a REST API. The following code sends a PUT request to update an existing todo with new data. Any data sent with a PUT request will completely replace the existing values of the todo.

You’ll use the same JSONPlaceholder endpoint you used with GET and POST, but this time you’ll append 10 to the end of the URL. This tells the REST API which todo you’d like to update:

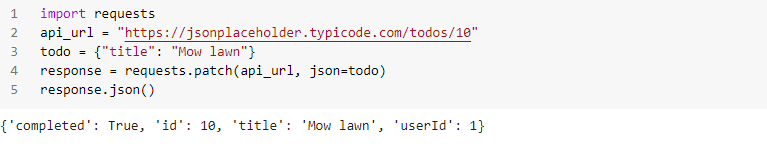


Here, you first call requests.get() to view the contents of the existing todo. Next, you call requests.put() with new JSON data to replace the existing to-do’s values. You can see the new values when you call response.json(). Successful PUT requests will always return 200 instead of 201 because you aren’t creating a new resource but just updating an existing one

# Phương thức Patch

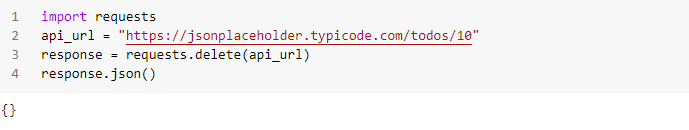
Next up, you’ll use requests.patch() to modify the value of a specific field on an existing todo. PATCH differs from PUT in that it doesn’t completely replace the existing resource. It only modifies the values set in the JSON sent with the request.

You’ll use the same todo from the last example to try out requests.patch(). Here are the current values:



# Phương thức delete

Last but not least, if you want to completely remove a resource, then you use DELETE. Here’s the code to remove a todo:



# Authentication



https://realpython.com/python-requests/

# JSON và XML

**Tài liệu tham khảo:**

1. <https://medium.com/geekculture/http-the-request-response-cycle-ddcb5bb70707>
2. <https://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf>
3. <https://xuanthulab.net/giao-thuc-http-va-cau-truc-co-ban-cua-http-message.html>
4. <https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httprequestmessage.version?view=net-5.0>
5. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>
6. <https://www.geeksforgeeks.org/python-urllib-module/>
7. <https://medium.com/geekculture/http-the-request-response-cycle-ddcb5bb70707>
8. <https://realpython.com/api-integration-in-python/>

|  |
| --- |
| BÀI TẬP CHƯƠNG 8 |

**[Yêu cầu sinh viên nộp bài]**

**Bài tập 1:** Sinh viên hãy hoàn thành thí dụ 1 bằng cách bổ sung vào đoạn chương trình để lưu kết quả hiện thị trong tập tin có định dạng html, có xử lý ngoại lệ **URLError** và **HTTPError**, và hãy nhận xét kết quả nhìn thấy được.

**Bài tập 2:** Sinh viên hãy cho biết kết quả của câu lệnh sau đây, sinh viên có nhận xét gì về kết quả này

x = urllib.request.urlopen('https://www.google.com/search?q=test')

**Bài tập 3:** Sinh viên hãy quan sát hai cách truy vấn dữ liệu từ trang <http://python.org> dưới đây, so sánh và nhận xét

**Cách 1:**

**import** **urllib.request**

url\_google = urllib.request.urlopen('http://python.org/')

html = url\_google.read()

**Cách 2:**

**import** **urllib.request**

**with** urllib.request.urlopen('http://python.org/') **as** response:

html = response.read()

**Bài tập 4:** Đoạn code dưới đây dùng để lưu trữ ảnh (**binary\_framing2.png**) lấy từ địa chỉ **https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages/binary\_framing2.png**, sinh viên hãy phát triển đoạn script này để có thể lưu hết toàn bộ các ảnh tại địa chỉ **https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages/**.

**import** **urllib.request**

urllib.request.urlretrieve("https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages/binary\_framing2.png", "binary\_framing2.png")

**Gợi ý**: Sinh viên sử dụng biểu thức regular-expession (thư viện re) để phân tích lấy url chứa ảnh.

**Bài tập 5:** Đoạn chương trình dưới đây đọc dữ liệu từ địa chỉ **'http://sixty-north.com/c/t.txt**' và kết quả lưu vào trong danh sách lstStory. Sinh viên hãy phát triển đoạn chương trình này để có thể lưu toàn bộ dữ liệu đọc được với định dạng tập tin lưu trữ là csv.

**from** **urllib.request** **import** **urlopen**

with urlopen('http://sixty-north.com/c/t.txt') as story:

lstStory = story.readlines()

print(lstStory[0])

print('done')

**Bài tập 6:**

**Phụ lục: Những Status Code thông dụng**

|  |  |  |
| --- | --- | --- |
| **Code** | **Status** | **Meaning** |
| 2xx Success | | |
| 200 | OK | Standard response for successful HTTP requests. The actual response will depend on the request method used. In a GET request, the response will contain an entity corresponding to the requested resource. In a POST request, the response will contain an entity describing or containing the result of the action. |
| 201 | Created | The request has been fulfilled, resulting in the creation of a new resource. |
| 204 | No Content | The server successfully processed the request and is not returning any content. |
| 206 | Partial Content | The server is delivering only part of the resource (byte serving) due to a range header sent by the client. The range header is used by HTTP clients to enable resuming of interrupted downloads, or split a download into multiple simultaneous streams. |
| 3xx Redirection | | |
| 301 | Moved Permanently | This and all future requests should be directed to the given URI.. |
| 302 | Found | The HTTP/1.0 specification (RFC 1945) required the client to perform a temporary redirect (the original describing phrase was "Moved Temporarily") |
| 303 | See Other | The response to the request can be found under another URI using a GET method. When received in response to a POST (or PUT/DELETE), the client should presume that the server has received the data and should issue a redirect with a separate GET message. |
| 4xx Client errors | | |
| 400 | Bad Request | The server cannot or will not process the request due to an apparent client error (e.g., malformed request syntax, size too large, invalid request message framing, or deceptive request routing). |
| 401 | Unauthorized | Similar to 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource. |
| 403 | Forbidden | The request was valid, but the server is refusing action. The user might not have the necessary permissions for a resource, or may need an account of some sort. |
| 404 | Not Found | The requested resource could not be found but may be available in the future. Subsequent requests by the client are permissible. |
| 429 | Too Many Requests | The user has sent too many requests in a given amount of time. |
| 5xx Server errors | | |
| 500 | Internal Server Error | A generic error message, given when an unexpected condition was encountered and no more specific message is suitable. |
| 502 | Bad Gateway | The server was acting as a gateway or proxy and received an invalid response from the upstream server |
| 503 | Service Unavailable | The server is currently unavailable (because it is overloaded or down for maintenance). Generally, this is a temporary state. |

1. https://en.wikipedia.org/wiki/Load\_balancing\_(computing) [↑](#endnote-ref-1)