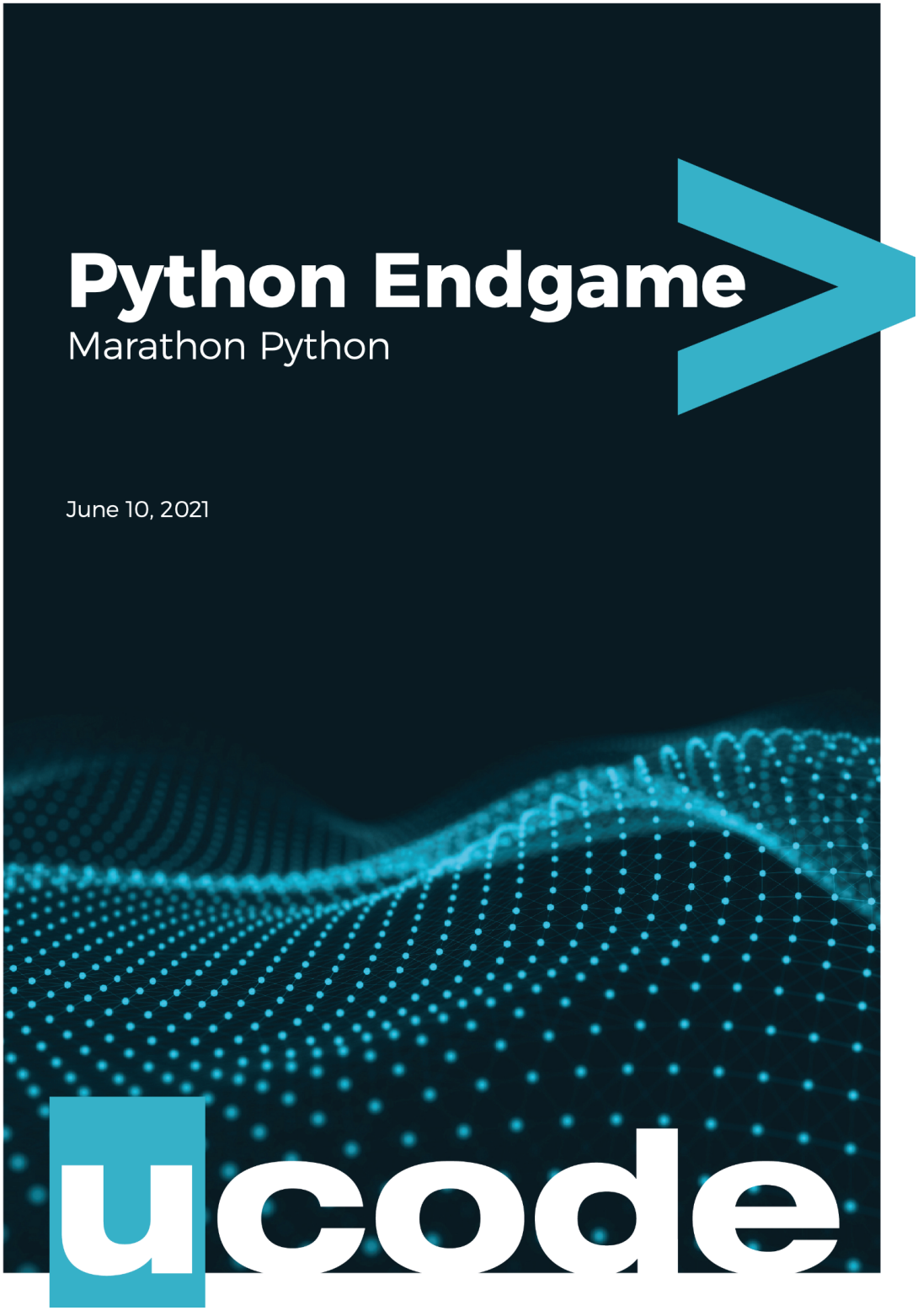

Python Endgame
Marathon Python
June 10, 2021
ucode

Contents

ucode

Engage

DESCRIPTION

Welcome to the final challenge!

Using an `API` (Application Programming Interface) is one of those "magic" skills that open up a world of new possibilities.

APIs provide a platform and environment for applications to enable them to communicate and understand each other. APIs define how the information that's passed between platforms is structured, so that applications can exchange information.

Using an API, you can access data such as weather information, sports scores, movie ratings, tweets, and more. You can also use an API to add functionality to your app.

Many APIs have thorough documentation that describes how to make requests to their endpoints and how to understand the responses. They also indicate the information and authentication required for a successful request.

Good luck and patience!

BIG IDEA

Interaction with an API.

ESSENTIAL QUESTION

How to get and process information from APIs?

CHALLENGE

Create an API client program with command-line and graphical interfaces.

Python Endgame | Marathon Python **> 2**

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- What is an API?
- What are APIs used for?
- What is an HTTP request?

- What parameters can an HTTP request consist of?
- How do HTTP requests work?
- What are the various HTTP request methods?
- What is the difference between GET and POST?
- What is the difference between POST, PUT, and PATCH?
- How to specify the query parameters when making a request with the requests library?
- What is an endpoint in the topic of API?
- What are status codes?
- Which status codes exist? What does the 404 code mean?
- What errors can occur when sending a request?
- What is REST? What are its principles?
- What is the difference between authorization and authentication?
- What is an API key?
- What are the most popular authentication methods for APIs?
- What are the pros and cons of using API keys?
- What are the pros and cons of using Basic Authentication?
- What tasks can the library argparse help with?
- How to write a great help screen for a program?
- What are your expectations for this challenge? What is your deadline?
- What tools will you use for teamwork? Jira, Asana, Trello, etc.?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Read the story.
- Meet with your team. Discuss teamwork organization and communication.

Python Endgame | Marathon Python **> 3**

- Learn more about APIs: what they are, why they are used and how. Also, you can read this article.
- Read about REST.
- Get to know the HTTPs methods and status codes.
- Here you can learn a little about the Python requests.Response Object.
- Watch a video about REST API concepts and examples.
- Investigate how to use the Python requests module with REST APIs.
- Read about the three most common API authentication methods and authentication with requests.
- Identify the strengths and expertise of each team member.
- Decide how to split the work in a way that is beneficial to each team member and the project in general. Listen to each other carefully.
- Clone your git repository issued on the challenge page in the LMS.
- Start to develop the solution. Offer improvements. Test your code.
- Explore new things. Talk, discuss and make conclusions.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- The challenge has to be carried out by the entire team.
- You are working as a team. In programming, knowledge sharing is very important, so don't be afraid to ask your team members for review.
- Each team member must understand the challenge and realization, and be able to reproduce it individually.
- It is your responsibility to assemble the whole team. Phone calls, SMS, messengers are good ways to stay in touch.
- You can proceed to `Act: Creative` only after you have completed all requirements in `Act: Basic`. But before you begin to complete the challenge, pay attention to the program's architecture. Take into account the fact that many features indicated in the `Act: Creative` require special architecture. And in order not to rewrite all the code somewhen, we recommend you initially determine what exactly you will do in the future.
- Be attentive to all statements of the story.
- Analyze all information you have collected during the preparation stages. Try to define the order of your actions.
- Perform only those tasks that are given in this document.
- Submit only those files that are described in the story. Only useful files allowed, garbage shall not pass!

ucode

Python Endgame | Marathon Python **> 4**

- Run the scripts using `python3`.
- Make sure that you have `Python` with a `3.8` version, or higher.
- Use the standard library available after installing `Python`. You may use additional packages/libraries that were not previously installed only if they are specified in the challenge description.
- To figure out what went wrong in your code, use `PEP 553 -- Built-in breakpoint()`.
- Complete the challenge according to the rules specified in the `PEP8 conventions`.
- The solution will be checked and graded by students like you. `Peer-to-Peer learning`.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.
- If you have any questions or don't understand something, ask other students or just Google it.

ucode

Python Endgame | Marathon Python **> 5**

Act: Basic

DIRECTORY

`./`

SUBMIT

`endgame.py, README.md, [any other files you need]`

General requirements

Create an API client program that can send HTTP requests to RESTful web-based APIs and display responses. The solution must store a history of requests and have both a command-line and a graphical user interfaces.

The very core functionality is the following:
make a request using an HTTP verb denoting the request method, a path to an API endpoint, and optionally some other information (like query parameters, a request body or headers, etc.), and display the response.
In addition to that, it must be possible to review the history of completed requests.

Base features:

- ability to send requests to APIs and display the responses
- command-line interface
- graphical user interface
- use of an SQL database to store request history
- use of the `argparse` library to parse command-line arguments
- logging to a file (appending, not overwriting)
- `endgame.py` as the main runnable script

Think about what information is important to log, and on what level. There are several requirements for logging throughout the story, but, for the most part, it's up to you (and so is the format of the logs).

The user of your program must be able to do the following (using either of the interfaces):

- build an API request
- send the request
- view the response in a desired format
- view the last ten requests made (and responses)
- clear request history
- hide personal information using variables

Python Endgame | Marathon Python **> 6**

Request

Use the library `requests` to send requests to web-based RESTful APIs. Make sure to handle all possible errors (incorrect URLs, not authorized, timeout, etc.) and log descriptive error messages on the `ERROR` level.

Your program must support the following request methods:

- `GET`
- `POST`
- `PUT`
- `PATCH`
- `DELETE`

The user must be able to specify various parameters before sending the request. Those parameters must include (but are not limited to):

- request method as an HTTP verb (e.g., `GET`) [required]
- endpoint URL (e.g., `https://api.hunter.io/v2/email-finder`) [required]
  Add URL validation (as you have done in `Sprint 04`)
- pairs of parameters to be sent in the URL's query string [optional]

  For example, the parameters `last_name=Lee` and `domain=gmail.com`, together with the endpoint mentioned above, will create the following request URL:
  `https://api.hunter.io/v2/email-finder?last_name=Lee&domain=gmail.com`
- username and password for `Basic Authentication` [optional]
- request headers [optional]
- request body [required for some methods]
- option of how to view the response body [optional] (raw, prettified JSON, or any other view your program supports)

Note: the tags 'required'/'optional' in the list above refer to whether the user of your program is required or not to specify the information for every request (not whether you are required or not to implement the feature).

If one or more of the required arguments are missing or invalid, do not attempt to send a request. Instead, display an informative error message.

Authentication

Most APIs require some sort of authentication. In many cases it is easy to send the credentials either with the query parameters or the headers. For example, API keys can be send in the URL's query string.

Implement additional support for `Basic Authentication` (requires username and password). The user must be able to specify their credentials for the request (as a separate command-line argument in CLI, or as a dedicated input area in GUI). Once you have the values, you will have to make sure the credentials are sent with the request.

Response

Upon receiving a response from your request (regardless of the interface type), the user must be able to see the response's:

- status code, such as `200` or `404`
- text representation of the status code, such as `'OK'` or `'Not Found'`
- response time (how much time has elapsed from sending the request to getting the response) in seconds (rounded to two decimal digits)
- body of the response (in the view specified by the user)

All of the above information can be pulled from the `Response` object. The display of the results must depend on the interface type (in the console for the CLI, within the graphic window for the GUI).

The only content-type of response body you are required to support is JSON. However, make sure to handle other cases without crashing.

In addition to displaying the results, log an `INFO` message about making the request (mentioning method and URL), and then log a message about the response (mentioning status code, method, and URL) to a logging level that makes sense with the status code.

History

Use an SQL database of your choice to store the history of requests. It must be possible to view the history and to clear it. Store the request information only if the request has actually received a response (including responses with unsuccessful status).

Required features for both interface types:

- Information on every request that receives a response must be stored in the database.
- For the last ten records, the user must be able to access that stored data.
- That information must include:
  - everything necessary for the user to recreate the request (all specified parameters)
  - status code of the response
  - body of the response

  This doesn't mean that you have to show all full information about the last ten requests at once. You could, for example, show a table or list with brief info, and then let the user select a request to see the full information.
- The user must be able to clear all records.

Regarding the requirement about the last ten requests: that is the minimum that you're required to handle. You are free to add more features, such as viewing a certain number of requests at a time, or all at once, etc.

The purpose of storing the history is to check what requests you have made, what were the results, and how to recreate the request. So, think carefully about how to store the records. Based on the stored record, the user must be able to see all the parameters they specified to make the request, and to see the received response (without making the request again).

Moreover, the records should be stored in the same way, regardless of whether the request was made within the GUI or the CLI. So if, for example, the user views the history using the CLI, they will see the requests made in both interfaces, not just the CLI. If you want, you can store information about the interface the request was made from, but it's not required.

CLI-specific instructions

Running the program without any arguments must display usage. If the program is called with `-h` or `--help` arguments, the program must display usage and a help screen.

The help screen must include:

- short (one or two sentences) description of the program
- the program's name and usage
- information on all available arguments (including a short yet informative help message for each)
- examples of how to use the program (at least five)

Most of the help screen is generated automatically by the argparse library. However, certain sections, such as argument help messages, have to be added by you.

Arguments

Your solution must support command-line arguments for the following purposes:

- general:
  - show help screen (generated automatically by argparse as the flag `-h` / `--help`)
  - launch GUI instead of CLI with the flag `--gui`

    The argument must store True when specified, and False otherwise. This is the only argument necessary to launch the GUI version of the program.

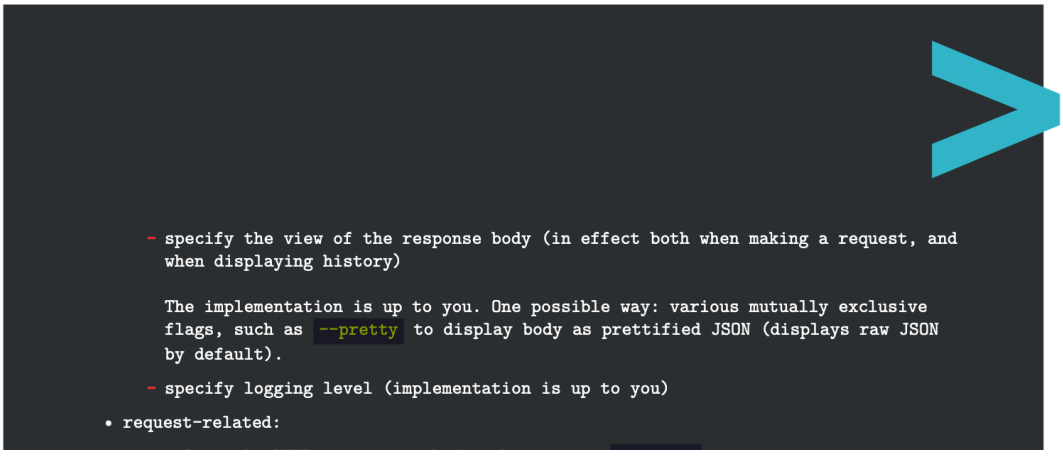
    Example: `python3 endgame.py --gui`

    In case the user specifies some other arguments as well, you have two options (choice is up to you):
    * ignore the extra arguments
    * autofill the corresponding input areas in the GUI with the values (e.g., if the the program was called like this: `python3 endgame.py --gui --method PUT`, PUT will become selected as the method)
  - show/clear history with argument `--history` and choices 'show' and 'clear'

    On `--history show`, the program must output information on the last ten requests. The information included must be enough to recreate the request (plus the status code of the response). The exact implementation is up to you, but here is one strategy:
    Print a table (with row indexes) of short information on the last ten requests, and a prompt to enter an index of a request (or a command to exit). If the user enters a valid index, the program prints full info on the request and the response body of the request at that index. You can see an example further in the section. To visualize the table, you may use any library you like (such as tabulate or PrettyTable).

    On `--history clear`, the program must clear the history, and display a corresponding message informing the user.

Python Endgame | Marathon Python **> 10**



  - specify the view of the response body (in effect both when making a request, and when displaying history)

    The implementation is up to you. One possible way: various mutually exclusive flags, such as `--pretty` to display body as prettified JSON (displays raw JSON by default).
  - specify logging level (implementation is up to you)
- request-related:

- select the HTTP request method with argument `--method`

  The argument must have the choices: 'GET', 'POST', 'PUT', 'PATCH', and 'DELETE'. Set the default value to 'GET'.

  Example: `python3 endgame.py [...] --method PUT [...]`

- specify the endpoint URL with argument `--endpoint` that takes a string

  Example: `python3 endgame.py [...] --endpoint https://reqres.in/api/users/2 [...]`

- specify query parameters with argument `--params` that takes any number of key-value pairs

  Hint: one of the ways to implement the key-value pair functionality is to set a flexible `nargs` and a custom `action` that parses the pairs.

  Example: `python3 endgame.py [...] --params country=Finland order=popular [...]`

- specify request headers with argument `--headers` that takes any number of key-value pairs

  Example: `python3 endgame.py [...] --headers user-agent=my-app/0.0.1 [...]`

- specify request body with argument `--body` that takes any number of key-value pairs

  Example: `python3 endgame.py [...] --body name=Alice age=50 [...]`

- specify username and password for Basic Authentication with argument `--auth` that takes exactly two arguments

  Example: `python3 endgame.py [...] --auth jerry 1234 [...]`

You are advised (but not required) to add short versions for the arguments, such as `-m` for `--method`, where possible.

Regarding the arguments that have a limited number of choices available: if the user has entered a value that is not among the accepted options, the program must display usage and an error message, such as:
"endgame.py: error: argument -m/--method: invalid choice: 'WRONG' (choose from 'GET', 'POST', 'PUT', 'PATCH', 'DELETE')" (argparse generates this error message automatically).

Python Endgame | Marathon Python **> 11**

Some examples of CLI use of the program (the variables are described later in the story):

```
>python3 endgame.py --method GET --endpoint {@curr_search} --params keywords=python
↪  category=technology language=en page_size=2 apiKey={@curr_key} --yaml
---Got response 200 OK in 2.27 seconds---
---Response body---
news:
- author: StackCommerce
  category:
  - technology
  description: Access over 350 lectures and 30 hours of content.
  id: aa957900-71de-4d82-bce9-972f18ea72b7
  image: https://mondrian.mashable.com/uploads%252Fstory%252Fthumbnail%252F128096%252F6
↪  7cb799b-723e-4d68-8432-7558727f2150.jpeg%252F640x360.jpeg?signature=YxRfGYbqyU_MZYB
↪  ATgKDNgQad18=&source=https%3A%2F%2Fblueprint-api-production.s3.amazonaws.com
  language: en
  published: 2021-06-08 04:00:00 +0000
  title: "Master the Python programming language for under \xA310"
  url: https://mashable.com/uk/shopping/june-8-python-bootcamp-master-course/
```


- author: Paul Krill
category:
- technology
description: "IBM\u2019s Uncertainty Qualification 360 is an open source library\
\ of Python algorithms for quantifying, estimating, and communicating the
↪ uncertainty\
\ of machine learning models. \n"
id: afda4cdc-0c54-4e16-97f3-2ee53dd6fa1c
image: https://images.techhive.com/images/article/2014/10/failure_defeat_inaccuracy_m
↪ istakes_aim_miss_target_dartboard_goal_aspiration_thinkstock_b-100476633-large.jpg
language: en
published: 2021-06-07 20:45:00 +0000
title: IBM Python toolkit measures AI uncertainty
url: https://www.infoworld.com/article/3620942/ibm-python-toolkit-measures-ai-uncerta
↪ inty.html
page: 1
status: ok
>python3 endgame.py --history show
---Request history---
.. Method URL Params Request body Status
0 GET {@curr_search} apiKey: '{@curr_key}' 200
category: technology
keywords: python
language: en
page_size: '2'
Enter request index to view full info, or "q" to quit: 0
---Request 0---
Python Endgame | Marathon Python > 12
ucode


..
Request info
Method
GET
URL
{@curr_search}
Params
apiKey: '{@curr_key}'
category: technology
keywords: python
language: en
page_size: '2'
Headers
Request body
Basic Authentication
Status
200
---Response---
{"status":"ok","news":[{"id":"aa957900-71de-4d82-bce9-972f18ea72b7","title":"Master the
↪ Python programming language for under \u00a310","description":"Access over 350
↪ lectures and 30 hours of content.","url":"https:\/\/mashable.com\/uk\/shopping\/jun
↪ e-8-python-bootcamp-master-course\/","author":"StackCommerce","image":"https:\/\/mo
↪ ndrian.mashable.com\/uploads%252Fstory%252Fthumbnail%252F128096%252F67cb799b-723e-4
↪ d68-8432-7558727f2150.jpeg%252F640x360.jpeg?signature=YxRfGYbqyU_MZYBATgKDNgQad18=&
↪ source=https%3A%2F%2Fblueprint-api-production.s3.amazonaws.com","language":"en","ca
↪ tegory":["technology"],"published":"2021-06-08 04:00:00
↪ +0000"},{"id":"afda4cdc-0c54-4e16-97f3-2ee53dd6fa1c","title":"IBM Python toolkit
↪ measures AI uncertainty","description":"IBM\u2019s Uncertainty Qualification 360 is
↪ an open source library of Python algorithms for quantifying, estimating, and
↪ communicating the uncertainty of machine learning models.
↪ \n","url":"https:\/\/www.infoworld.com\/article\/3620942\/ibm-python-toolkit-measur
↪ es-ai-uncertainty.html","author":"Paul

```
↪  Krill","image":"https:\/\/images.techhive.com\/images\/article\/2014\/10\/failure_d
↪  efeat_inaccuracy_mistakes_aim_miss_target_dartboard_goal_aspiration_thinkstock_b-10
↪  0476633-large.jpg","language":"en","category":["technology"],"published":"2021-06-0
↪  7 20:45:00
↪  +0000"}],"page":1}
>python3 endgame.py --method PUT --endpoint http://jsonplaceholder.typicode.com/posts/9
↪  --body id=9 title="My title"
---Got response 200 OK in 0.32 seconds---
---Response body---
{
  "id": 9,
  "title": "My title"
}
>
```

ucode

Python Endgame | Marathon Python **> 13**

History:

```
>python endgame.py --history show --yaml
---Request history---
====  =======  =========================  ==============  ================  =======
  ..  Method   URL                        Params          Request body       Status
====  =======  =========================  ==============  ================  =======
   0  POST     {@reqres}users                             name: Matilda         201
                                                          super power: fly
   1  GET      https://api.github.com/user                                      200
   2  GET      {@trivia}                  amount: '1'                           200
                                          category: '11'
                                          type: multiple
   3  GET      {@reqres}users                                                   200
====  =======  =========================  ==============  ================  =======
Enter request index to view full info, or "q" to quit: 2
---Request 2---
====================  ==============
..                    Request info
====================  ==============
Method                GET
URL                   {@trivia}
Params                amount: '1'
                      category: '11'
                      type: multiple
Headers
Request body
Basic Authentication
Status                200
====================  ==============
---Response---
response_code: 0
results:
- category: 'Entertainment: Film'
  correct_answer: Boomstick
  difficulty: easy
  incorrect_answers:
  - Bloomstick
```

```
  - Blastbranch
  - 2-Pump Chump
  question: In the 1992 film &quot;Army of Darkness&quot;, what name does Ash give
    to his double-barreled shotgun?
  type: multiple
>python endgame.py --history clear
---Request history cleared---
>
```

GUI-specific instructions

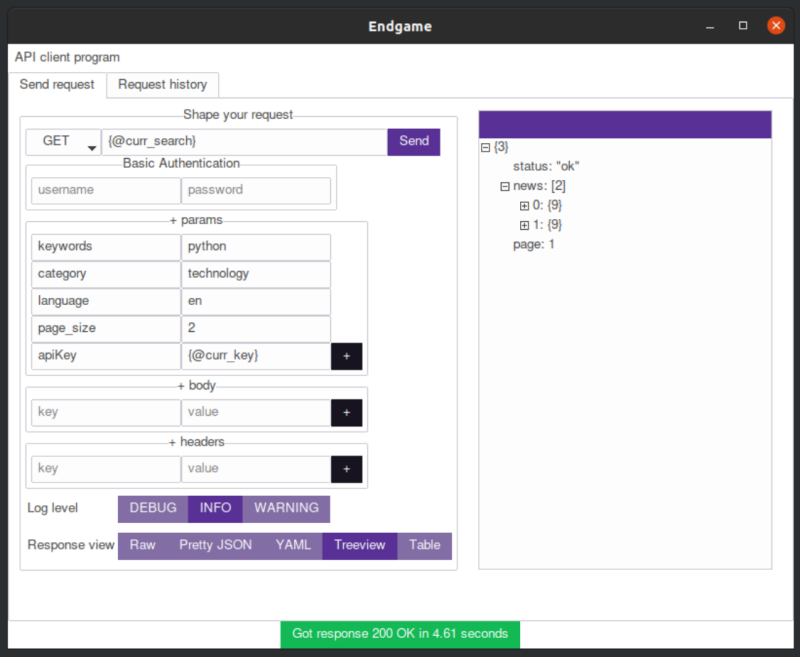
Your program must have a GUI. On launch, a window must open. All input parameters and the results must be within that GUI window, not in the console. Use Tkinter as the GUI framework for your program. You're allowed to use Tkinter's additional modules.

The GUI version of the solution must not require any additional command-line arguments to start (apart from the argument that triggers GUI launch). All the information that is required to make the request must be configurable within the GUI.

The user must be able to:

- select the logging level
- construct a request by editing various fields:
  - select the HTTP method from multiple choices
  - enter the endpoint URL
  - add query parameters as key-value pairs
  - add request body as key-value pairs
  - add request headers as key-value pairs
  - enter the username and password for Basic Authentication
  - select a format option for the view of the resulting response body
- send the constructed request
- view the response in the format specified
- view the history of at least the last ten requests
- clear history

Here is an example of how your GUI may look like:



Variables

In order to make your program a little more secure to use, and to save the user's time, implement variables that will hold various string data. Before you get started, read about how Postman uses variables (here).

How and where exactly to store the variables is up to you. However, the user must be able to add/remove/edit the variables without looking into your code. The easiest way that we recommend is to have a JSON or YAML file to store the variables. Below is an example of such a file with some data (credentials are obviously fake):

```
reqres: https://reqres.in/api/
reqres_key: 68eafa27-c6d3-415d-9473-142983f6967b
movies_name: frodo
movies_psw: myPrecious123
spotify_key: a11b0607-e1f6-455b-bce6-f5131f42f277
```

To use a variable, the user would have to type the name of the variable surrounded by some special characters. Here are several examples using '{@' and '}' as the special delimiters.

```
python3 endgame.py -m GET -e {@reqres}users -p apiKey={@reqres_key}
```

```
python3 endgame.py -m GET -e https://api.fakedb.org/movie/ -auth {@movies_name} {@movies_psw}
```

The characters you use to separate the variables are up to you. However, make sure they don't cause errors in the command line (some characters have a special meaning).

The variables must be supported in both CLI and GUI. You are required to handle the variables in the endpoint, query parameters, headers, and authentication. Adding variable functionality for other data is up to you.

README

Write a comprehensive README for your solution. You might want to incorporate some parts of the help screen into it.

Your README.md file must contain the following sections:

- Name
  The name of your project.
- Description
  A short paragraph describing your project.
- Prerequisites
  List of what needs to be installed/set up (and what version) for your program to run, and how to do it.
- Usage
  How to run your program, what arguments it may take. Show console examples of all the options.
  Include screenshots of your GUI in different scenarios.
- Authors
  List the people who have worked on the project and in what roles. Add links to their profiles on relevant sites.

APIs for testing

Below you can find a list of some APIs that you can use to test your solution:

- REQRES
  A real API with fake data to practice making requests. Provides endpoints to test different methods and scenarios (delayed response, unsuccessful POST, etc.). Read the documentation here. Does not require any authentication.

- The Open Trivia Database
  A simple API that generates trivia questions. Read the documentation here. Does not require any authentication.
- Where the ISS at?
  An API that provides information on the positions of the ISS. Read the documentation here. Does not require any authentication.
- Currents
  News API with some free features. Read the documentation here. Requires an API key.
- GitHub
  GitHub API. Get started here. Can be used for basic GET requests without any authentication, but also available with multiple methods of authentication, including Basic Authentication.

To find more public APIs, we recommend looking at this list.

Make sure to read reference documentation before testing endpoints of a particular API.
Pay attention to authentication requirements and rate limits.

Act: Creative

DESCRIPTION

It is the place where your imagination and creativity plays a major role. Implement additional features to make the program better and more unique. Listed below are a few ideas you can add to your program. You can come up with everything you want to improve your program. Creative features:

- support for various content formats of request body and response body
- functionality to add credentials for other authentication systems (e.g., OAuth 2.0)
- description field for a request (for more informative request history)
- creating endpoint tests
- hashing of the variables (the variables have to be editable through the program)
- pulling command-line arguments from a file (like this: `python3 endgame.py #arguments.txt`)

- request history features:
  - filtering or separate tabs/sections of requests by a characteristic (success, request method, etc.)
  - possible to see all records + pagination/scrollbar
  - possible to specify how many records to see
  - ability to select a request from the history and send it again
  - saving requests to favorites
  - deleting a particular record
- other creative features

ucode

Python Endgame | Marathon Python **> 20**

Evaluation

What you need for assessment

- Solution on your git repository
- Presentation of your product

Presentation is an important stage in the process of product creation.
You must be able to defend your solution in front of an audience.
First impressions are critical.
Proper preparation is vital to presenting your product in the best light possible.

The objective of product presentation can be different depending upon the target audience and the presentation should be adjusted accordingly. It is important to know your audience and why they are interested enough to hear your presentation.

You will need to create a presentation that you will use during the defense, for other students. The presentation can be in any form convenient to you.

Examples of presentation tools

- Google Slides
- Keynote
- Prezi
- YouTube or any other video-sharing service

In your presentation you should tell about the experience of working on the product. From the moment you put together the team, up to, in fact, the stage of creating the solution and preparing for the presentation. You must present your product as if you want to sell it to investors who are interested in making a huge investment in the development of your product.

The product is one of the most important keywords here. Place a small piece demonstrating what you've done in your presentation. This way, the audience will see not only your brilliant presentation skills, but also some proof of your hard work.

The optimal time of the entire presentation is 7-10 minutes. It is pretty enough to make the audience interested in your product, show all the necessary features, and not to overload them.

Discuss with the team how you will present your solution. It is very important to prepare for a good presentation. So decide how you will divide the presentation content among all of your team members.

A bunch of advices

- Make a quick introduction of your product. Tell about your team, idea of your solution, why you chose it.
- Tell how you identified the solution. What kind of feedback have you obtained about the selected product topic, what were the expectations of the users, how did you take into account user feedback?

- It will be interesting if you tell how you worked on the solution. What influenced you while working on the creation of the product. What purpose did you follow?
- Tell what techniques and algorithms did you use? How did you use it? Why such a choice? What difficulties did you encounter during its development?
- Don't forget to show your product in action. Record the screen of how you use your program. Make it look like a trailer of your app. To pay attention to the most essential moments of your solution, capture the screen and use it in your presentation. Remember that you are limited in time.
- Describe how you see the further development of the product. What did you not have time to implement? How would you like to improve your solution? How would you develop your product if you received an investment in it?
- Try to save time for Q&A from the audience.

Examples of resources to prepare for the presentation

- How to create an awesome product presentation
- How to prepare and give a speech
- How to mentally prepare for a speech
- 7 Amazing sales presentation examples (and how to make them your own)
- 8 secrets of successful presentation (with examples)

ucode

Python Endgame | Marathon Python **> 22**

Reflection

The next stage of your work is reflection. During the reflection, you are doing a retrospective of all work on the product. This allows you to understand the experience gained, consolidate your knowledge, and understand how to overcome the difficulties in the future.
Do not delay, gather the next day after the presentation, and discuss your experience and what you have learned while working on the product. Make a brainstorm and reflection on the results of this challenge. Take note of the topics discussed.

Example of topics

- Discuss all stages of your product. From team building to product presentation.
- Discuss your teamwork, interaction in the team. What were the issues of your teamwork? How can they be solved? What did you like the most in your team organization process? Did you use any approach to team management and software development?
- Discuss the competencies of each team member. Discuss positive and negative points. What expertise did you improve? Be open and truthful. This will help all of you to become better in the future.
- Discuss the technical implementation of your solution. What mistakes did you make while developing your product? How can you avoid them in the future? What new knowledge have you gained while working with selected technologies? What will you use again upon development?
- Discuss testing and user feedback. What kind of improvements would you make? What mistakes were made in the process? What is worth paying more attention to? What was the most attractive fo users? How could you do it much better?
- Discuss the preparation for the presentation. What did you not pay attention to? What will you take into account in the future? What part of the preparation took a lot of time?
- Discuss the presentation of your product. What you have not considered? What were your advantages? What could help you get your presentation better?
- Discuss how you can share your experience with the world. It will consolidate your knowledge and experience, so take it seriously.

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.