Training materials

- Ensembl training materials are protected by a CC BY license [CC BY](http://creativecommons.org/licenses/by/4.0/)
- If you wish to re-use these materials, please credit Ensembl for their creation
- If you use Ensembl for your work, please cite our papers
- [http://www.ensembl.org/info/about/publications.html](http://www.ensembl.org/info/about/publications.html)
Ensembl REST API course

Ben Moore
Course agenda

• Ensembl and the gene model
• What is REST
• Ensembl REST server features
• Fetching a single endpoint
• Decoding the response to link together endpoints
• POST endpoints
• Rate limiting
Course materials

http://training.ensembl.org/events/

• Slides
• Notebooks in Python and R
  • Use whichever notebook you feel comfortable with
  • You will need to clone it with your Microsoft Account
• When we demo the example answers we will use Python only
Ensembl Features

- Gene builds for >200 species
- Gene trees
- Regulatory build (ENCODEx)
- Variation display and VEP
- Display of user data
- BioMart (data export)
- Programmatic access via the APIs
- Completely Open Source
Vertebrate species on Ensembl

www.ensembl.org
Non-vertebrates on Ensembl genomes

Bacteria
- Escherichia coli
- Staphylococcus aureus

Protists
- Plasmodium falciparum
- Toxoplasma gondii

Fungi
- Saccharomyces cerevisiae
- Aspergillus fumigatus

Metazoa
- Drosophila melanogaster
- Homo sapiens

Plants
- Arabidopsis thaliana
- Oryza sativa

www.ensemblgenomes.org
# Ensembl and Ensembl Genomes

<table>
<thead>
<tr>
<th>Released</th>
<th>Ensembl</th>
<th>Ensembl Genomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td></td>
<td>2009</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Species</th>
<th>Ensembl</th>
<th>Ensembl Genomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertebrates (fly, worm and yeast as outgroups)</td>
<td></td>
<td>Non-vertebrates (protists, plants, fungi, metazoa, bacteria)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Ensembl</th>
<th>Ensembl Genomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>by Ensembl</td>
<td></td>
<td>in collaboration with the scientific communities</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL</th>
<th>Ensembl</th>
<th>Ensembl Genomes</th>
</tr>
</thead>
<tbody>
<tr>
<td>rest.ensembl.org</td>
<td></td>
<td>rest.ensembl.org</td>
</tr>
</tbody>
</table>
Human genome assemblies

- **GRCh38 (aka hg38)**
  - No gaps. Many rare/private alleles replaced.
  - [rest.ensembl.org](rest.ensembl.org)
  - Software regularly updated
  - Data regularly updated

- **GRCh37 (aka hg19)**
  - 250 gaps
  - [grch37.rest.ensembl.org](grch37.rest.ensembl.org)
  - Software regularly updated
  - Data only rarely updated
Release cycle

- New/updated REST endpoints
- Updated regulation data
- Updated variation data
- Comparison on new genes and genomes
- New genome assemblies
- Underlying software updates
- Updated gene sets

2-3 months

Oct 2022 to Jan 2023
REST Archives

Starting with release 87, there are REST archives (GRCh38 only). We will continue to provide archive services for up to five years, to match the Ensembl website archives.

http://e87.rest.ensembl.org
Ensembl Data Model
Ensembl Data Model

Primary feature types of Genes, Transcript, and Exons

A Gene is a set of alternatively spliced Transcripts

A Transcript is a set of Exons
Ensembl Data Model

Translations are not Features.

A Translation object defines the UTR and CDS of a Transcript.

Peptides are not stored in the database, they are computed on the fly using Transcript objects.

Not all transcripts have a translation (e.g. ncRNAs)
Features

Features have a defined location on the genome

Start and end are always plotted on the forward strand

start < end
What is a REST API?

REpresentational State Transfer. It describes how one system can communicate state with another.

Typically over HTTP(S), providing a machine readable, language agnostic method to access remote data or services.

http://rest.ensembl.org/datalwant

Gene: IRAK4, start: 43758944, end: 43789543, ...
Ensembl REST

- Language agnostic access to Ensembl datasets
- Only a fraction of the functionality of the Perl API is exposed

http://rest.ensembl.org
What Ensembl REST is and is not

+ HTTP access to Ensembl data
+ Stable service
+ Limited by network latency
+ Read only
+ Versioned with archives

- No mirrors
- Not an efficient data mining solution
- Incomplete coverage
What is an endpoint?

“In REST, the resource typically refers to some object or set of objects that are exposed at an API endpoint. /api/users/johnny. An endpoint by itself is just a reference to a uri that accepts web requests that may or may not be RESTful. /services/service.asmx.”

An endpoint is a particular output that you can get given a particular input.

It is a function that interacts with our database.
Endpoint documentation

Full documentation of all the endpoints is found at:

http://rest.ensembl.org

The documentation lists:

• All the endpoints grouped by function
• The required parameters for each endpoint
• Optional parameters
• Example code for using the endpoints
Functional groupings

- Archive
- Comparative Genomics
- Cross References
- Information
- Lookup
- Mapping
- Ontology & Taxonomy
- Sequence
- Variation, etc...

Ensembl REST API Endpoints

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET archival/sid</td>
<td>Uses the given identifier to return the archived sequence</td>
</tr>
<tr>
<td>POST archival</td>
<td>Retrieve the archived sequence for a set of identifiers</td>
</tr>
</tbody>
</table>

Comparative Genomics

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET gene/eqsid</td>
<td>Retrieves a gene tree for a gene tree stable identifier</td>
</tr>
<tr>
<td>GET gene/mem/id</td>
<td>Retrieves the gene tree that contains the gene / transcript / translation stable identifier</td>
</tr>
<tr>
<td>GET gene/symb/symb</td>
<td>Retrieves the gene tree that contains the gene identified by a symbol</td>
</tr>
<tr>
<td>GET align/region/symb/region</td>
<td>Retrieves genomic alignments as separate blocks based on a region and species</td>
</tr>
<tr>
<td>GET homolog/sid</td>
<td>Retrieves homology information (ortholog) by Ensembl gene id</td>
</tr>
<tr>
<td>GET homolog/symb/symb</td>
<td>Retrieves homology information (ortholog) by symbol</td>
</tr>
</tbody>
</table>

Cross References

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET symbol/symbol/symbol</td>
<td>Looks up an external symbol and returns all Ensembl objects linked to it. This can be a display name for a gene, transcript, an alias, or an externally linked reference. If a gene's transcript is linked to the supplied symbol, the service returns both gene and transcript (it supports transcript links).</td>
</tr>
<tr>
<td>GET symbol/symbol</td>
<td>Performs lookups of Ensembl identifiers and retrieves their external references in other databases</td>
</tr>
<tr>
<td>GET symbol/symbol</td>
<td>Performs a lookup based on the primary accession or display label of an external reference and returning the information</td>
</tr>
</tbody>
</table>

EMBL-EBI
Endpoint Documentation

**GET lookup/id/:id**

Find the species and database for a single identifier e.g. gene, transcript, protein

### Parameters

#### Required

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>An Ensembl stable ID</td>
<td></td>
<td>ENSG00000157764</td>
</tr>
</tbody>
</table>

#### Optional

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Default</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>String</td>
<td>Name of the callback subroutine to be returned by the requested JSONP response. Required ONLY when using JSONP as the serialisation method. Please see the user guide.</td>
<td>-</td>
<td>randomlygeneratedname</td>
</tr>
<tr>
<td>db_type</td>
<td>String</td>
<td>Restrict the search to a database other than the default. Useful if you need to use a DB other than core</td>
<td>-</td>
<td>core otherfeatures</td>
</tr>
<tr>
<td>expand</td>
<td>Boolean</td>
<td>Expands the search to include any connected features, e.g. if the object is a gene, its transcripts, translations and exons will be returned as well.</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

You must include the id in the URL in this position

You can choose to include these in the URL in the format: `parameter=option`
Sample Code

```perl
use strict;
use warnings;

use HTTP::Tiny;

my $http = HTTP::Tiny->new();

my $server = 'http://rest.ensembl.org';
my $ext = '/lookup/id/ENSG00000157764?expand=1';
my $response = $http->get($server.$ext,
  headers => { 'Content-type' => 'application/json' });

die "Failed!\n" unless $response->[success];

use JSON;
use Data::Dumper;
if(length $response->[content]) {
```
Making a REST call in the browser

- The easiest way to make REST calls is to put URLs into the browser
- This can be used as a quick look-up
- This can help you to test the URLs in your scripts to see:
  - If they work
  - If you’ve included the correct parameters
  - What the output looks like
Pinging the database

Ping confirms that you have a connection to the database

http://rest.ensembl.org/info/ping?content-type=application/json

{  
    ping: 1
}


Requesting a gene by ID

http://rest.ensembl.org/lookup/id/ENSG00000157764?content-type=application/json

```json
{
  "source": "ensembl_havana",
  "object_type": "Gene",
  "logic_name": "ensembl_havana_gene",
  "version": 12,
  "species": "homo_sapiens",
  "description": "B-Raf proto-oncogene, serine/threonine kinase [Source:HGNC Symbol;Acc:HGNC:1097]",
  "display_name": "BRAF",
  "assembly_name": "GRCh38",
  "biotype": "protein_coding",
  "end": 140924764,
  "seq_region_name": "7",
  "db_type": "core",
  "strand": -1,
  "id": "ENSG00000157764",
  "start": 140719327
}
```
HTTP Status Codes

The server uses HTTP status codes to signal the request outcome

http://rest.ensembl.org/thisdoesntexist
## HTTP Status Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
<td>Request was a success</td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
<td>Occurs during exceptional circumstances such as the service is unable to find an ID. Check if the response Content-type or Accept was JSON. If so the JSON object is an exception hash with the message keyed under error</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>You are submitting far too many requests and have been temporarily forbidden access to the service. Wait and retry with a maximum of 15 requests per second.</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
<td>Indicates a badly formatted request. Check your URL</td>
</tr>
</tbody>
</table>

https://github.com/Ensembl/ensembl-rest/wiki/HTTP-Response-Codes
## HTTP Status Codes (cont.)

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>408</td>
<td>Timeout</td>
<td>The request was not processed in time. Wait and retry later</td>
</tr>
<tr>
<td>429</td>
<td>Too Many Requests</td>
<td>You have been rate-limited; wait and retry. The headers X-RateLimit-Reset, X-RateLimit-Limit and X-RateLimit-Remaining will inform you of how long you have until your limit is reset and what that limit was. If you get this response and have not exceeded your limit then check if you have made too many requests per second.</td>
</tr>
<tr>
<td>503</td>
<td>Service Unavailable</td>
<td>The service is temporarily down; retry after a pause</td>
</tr>
<tr>
<td>418</td>
<td>I’m a teapot</td>
<td>An April Fools joke added in 1998, who said computer scientists don’t have a sense of humour?</td>
</tr>
</tbody>
</table>

https://github.com/Ensembl/ensembl-rest/wiki/HTTP-Response-Codes
Exercises 1

1. Find an endpoint which you can use to lookup information about a gene using its symbol.
2. Create a URL to find information about the gene *ESPN* in human.
3. Expand your results to include information about transcripts.
Answers 1

1. http://rest.ensembl.org/documentation/info/symbol_lookup
Scripting around REST API calls

Scripting around calls allows you to:

- Extract specific bits of data from your REST call.
- Output in your preferred format.
- Link together calls for more complicated queries.
- Integrate your queries into a larger pipeline.
Language agnostic access

- REST APIs are designed to be accessed using any programming language.
- Calls can be made and decoded within any script.
- We have examples in Python, Perl and R.
Python modules

- To make requests in Python, you will need the `requests` package:
  - http://docs.python-requests.org/en/master/user/install/ (not needed for this course, this is all set up in your Python Notebook)
- To decode JSON you will need the `JSON` package:
  - Should ship with standard Python installations
- You’ll need `pprint` to print JSON in an easy to read way
  - Should ship with standard Python installations

```python
import requests, sys, json
from pprint import pprint
```
R libraries

• To make requests in R you will need the `httr` library
• To decode JSON you’ll need the `jsonlite` package

```r
library(httr)
library(jsonlite)
```
Requesting a gene by ID

http://rest.ensembl.org/lookup/id/ENSG00000157764?content-type=application/json

{
    "source": "ensembl_havana",
    "object_type": "Gene",
    "logic_name": "ensembl_havana_gene",
    "version": 12,
    "species": "homo_sapiens",
    "description": "B-Raf proto-oncogene, serine/threonine kinase [Source:HGNC Symbol;Acc:HGNC:1097]",
    "display_name": "BRAF",
    "assembly_name": "GRCh38",
    "biotype": "protein_coding",
    "end": 140924764,
    "seq_region_name": "7",
    "db_type": "core",
    "strand": -1,
    "id": "ENSG00000157764",
    "start": 140719327
}
Making a request – Python

• Make a string of the server (you’ll use this multiple times)
• Make another string of the extension with all the parameters

```python
import requests, sys

server = "http://rest.ensembl.org"
ext = "/lookup/id/ENSG00000157764?expand=1"

r = requests.get(server+ext, headers={ "Accept" : "application/json"})

pprint (r)
```
Making a request – R

- Make a string of the server (you’ll use this multiple times)
- Make another string of the extension with all the parameters

```r
library(httr)
library(jsonlite)

server <- "http://rest.ensembl.org"
ext <- "/lookup/id/ENSG00000157764"

r <- GET(paste(server, ext, sep = ""), accept("application/json"))

r
```
Error handling – Python

You should never assume a response will return correctly.

```python
import requests, sys

server = "http://rest.ensembl.org"
ext = "/lookup/id/ENSG00000157764?expand=1"

r = requests.get(server+ext, headers={ "Accept" : "application/json"})

if not r.ok:
    r.raise_for_status()
```

Check the response code returned by the server.
Error handling – R

You should never assume a response will return correctly.

```r
library(httr)
library(jsonlite)

server <- "http://rest.ensembl.org"
ext <- "/lookup/id/ENSG00000157764"

r <- GET(paste(server, ext, sep = ""), content_type("application/json"))

r

stop_for_status(r)
```

Check the response code returned by the server.
HTTP allows the serving of different representations of a resource based on client preferences.

Content-type and Accept headers are how servers and clients negotiate what format they will communicate with.

*text/html, text/plain, application/json, image/png, etc.*
Accept

- The returned content types can be specified in the header as accept (you’ll need to use content-type in URLs)
- Endpoint documentation pages list allowed content-types

https://github.com/Ensembl/ensembl-rest/wiki/Output-formats
Decoding the response – Python

- In most cases you’ll be using JSON formatted responses
- Most languages have JSON parsers that return the data as a structure
- In Python pretty print (`pprint`) will give you a human readable format

```python
decoded = r.json()

pprint(decoded)
```
Decoding the response – R

- In most cases you’ll be using JSON formatted responses
- Most languages have JSON parsers that return the data as a structure
- In R `pretty` will give you a human readable format

```r
decoded <- content(r, "text")
pretty(decoded)
```
Decoding JSON

http://rest.ensembl.org/lookup/id/ENSG00000157764?content-type=application/json

```json
{
   "source": "ensembl_havana",
   "object_type": "Gene",
   "logic_name": "ensembl_havana_gene",
   "version": 12,
   "species": "homo_sapiens",
   "description": "B-Raf proto-oncogene, serine/threonine kinase [Source:HGNC Symbol;Acc:HGNC:1097]",
   "display_name": "BRAF",
   "assembly_name": "GRCh38",
   "biotype": "protein_coding",
   "end": 140924764,
   "seq_region_name": "7",
   "db_type": "core",
   "strand": -1,
   "id": "ENSG00000157764",
   "start": 140719327
}
```
Decoding JSON

- JSON is essentially a massive dictionary/hash/dataframe with keys and values.
- Sometimes a key may then contain another nested dictionary or list
  - Which may contain another
    - And another
      - And another
- Look at the json to work out what keys you need
- You can cycle through all keys in a dictionary with for loops
Helper function

• The helper function in your python script makes your life easier by:
  • Calling the request with the specified server, extension and content type.
  • Getting the status of a failed query
  • Decoding the JSON (if you’ve used JSON as your content type)
  • Returning the text (if you use any other content type)
• Add it to every script then just call it when you need to fetch an endpoint
def fetch_endpoint(server, request, content_type):
    """
    Fetch an endpoint from the server, allow overriding of default content-type
    """
    r = requests.get(server+request, headers={ "Accept" : content_type})

    if not r.ok:
        r.raise_for_status()
        sys.exit()

    if content_type == 'application/json':
        return r.json()
    else:
        return r.text
R Helper function

```r
Fetch_endpoint <- function(server, request, content_type) {
  ##
  # Fetch an endpoint from the server, allow overriding of default content-type
  ##
  r <- GET(paste(server, request, sep = ""), accept(content_type))
  stop_for_status(r)

  if (content_type == 'application/json') {
    return (fromJSON(content(r, "text")))
  } else {
    return (content(r, "text"))
  }
}
```
Exercises 2

1. Write a script to lookup the gene called *ESPN* in human and print the results in JSON.
Using results

Since JSON is a dictionary, you can pull out a single datapoint using the key.

```json
{
    "source": "ensembl_havana",
    "object_type": "Gene",
    "logic_name": "ensembl_havana_gene",
    "version": 12,
    "species": "homo_sapiens",
    "description": "B-Raf proto-oncogene, serine/threonine kinase [Source:HGNC Symbol;Acc:HGNC:1097]",
    "display_name": "BRAF",
    "assembly_name": "GRCh38",
    "biotype": "protein_coding",
    "end": 140924764,
    "seq_region_name": "7",
    "db_type": "core",
    "strand": -1,
    "id": "ENSG00000157764",
    "start": 140719327
}
```
Since JSON is a dictionary, you can pull out a single datapoint using the key.

```python
server = "http://rest.ensembl.org/
ext = "lookup/id/ENSG00000157764?"
con = "application/json"
get_gene = fetch_endpoint(server, ext, con)

symbol = get_gene['display_name']
print (symbol)
```
Using results – R

Since JSON is a dataframe, you can pull out a single datapoint using the key.

```r
server <- "http://rest.ensembl.org/
ext <- "lookup/id/ENSG00000157764?"
con <- "application/json"
get_gene <- fetch_endpoint(server, ext, con)

symbol <- get_gene$display_name
symbol
```
Nested JSON lists

http://rest.ensembl.org/overlap/region/human/7:140424943-140444564?feature=gene;content-type=application/json

```
[{
  "gene_id":"ENSG00000146955",
  "Feature_type":"gene",
  "external_name":"RAB19",
  "description":"RAB19, member RAS oncogene family [Source:HGNC Symbol;Acc:HGNC:19982]",
  "Biotype":"protein_coding",
  "id":"ENSG00000146955",
},
{
  "gene_id":"ENSG00000103200",
  "Feature_type":"gene",
  "external_name":"AC069335.1",
  "Description":null,
  "Biotype":"processed_pseudogene",
  "id":"ENSG00000103200"
}]
```

List delineated by square brackets [ ] – no keys
Dictionary delineated by curly brackets { } – key-value pairs
Exercises 3

1. Write a script to lookup the gene called *ESPN* in human and print the stable ID of this gene.

2. Get all variants that are associated with the phenotype 'Coffee consumption'. For each variant print
   a. the p-value for the association
   b. the PMID for the publication which describes the association between that variant and ‘Coffee consumption’
   c. the risk allele and the associated gene.

3. Get the mouse homologue of the human *BRCA2* and print the ID and sequence of both.
Other content types – Python

• If you specify another content type (not JSON), the helper function will get you this as text
• This can be used to get:
  • Sequence in FASTA
  • Gene trees and homologues in various formats
  • Alignments

```python
if content_type == 'application/json':
    return r.json()
else:
    return r.text
```
Other content types – R

• If you specify another content type (not JSON), the helper function will get you this as text
• This can be used to get:
  • Sequence in FASTA
  • Gene trees and homologues in various formats
  • Alignments

```r
if (content_type == 'application/json'){
  return (fromJSON(content(r, "text")))
} else {
  return (content(r, "text"))
}
```
Other content types

- Endpoint documentation pages list allowed content-types
- The wiki lists how you specify these

https://github.com/Ensembl/ensembl-rest/wiki/Output-formats
Exercises 4

1. Get the gene tree predicted for the gene ENSG00000189221 in full nh format.
2. Get the sequence of the gene ENSG00000157764 in FASTA.
Linking endpoints together

• If you can pull a datapoint from the JSON, you can use it as input for another endpoint.
• You’ll need to link objects and extensions together.
Exercises 5

1. Using the script from 3.1, add a call to fetch and print the sequence for the gene ESPN in FASTA.

2. Print the stable ID of any regulatory features that overlap the region 1000 bp upstream of the ESPN gene. (Hints: get the gene ID first, then check the strand of the gene to see which way is upstream.)
Features

Features have a defined location on the genome

Start and end are always plotted on the forward strand

start < end
HTTP Methods - GET vs POST

GET http://rest.ensembl.org/lookup/ENSG00000157764

POST http://rest.ensembl.org/lookup/

{ "ids" : ["ENSG00000157764", "ENSG00000248378"]}
Using POST - Python

```python
import requests, sys

server = "http://rest.ensembl.org"
ext = "/lookup/id"
headers={ "Content-Type" : "application/json", "Accept" : "application/json"}
r = requests.post(server+ext, headers=headers, data='{
  "ids": ["ENSG00000157764", "ENSG00000248378"
```}

# error checking removed for space

decoded = r.json()
pprint (decoded)
```
library(httr)
library(jsonlite)

server <- "http://rest.ensembl.org"
ext <- "/lookup/id"
genes <- c("ENSG00000157764", "ENSG00000248378")
body_values <- toJSON(list(ids=genes))

r <- POST(paste(server, ext, sep = ""), content_type("application/json"), accept("application/json"), body = body_values)

prettify(content(r, "text"))
Helper function

- You can also have a helper function for POST queries
- You’ll need to create a list of your values
- If you have a Python list you can convert it to a JSON list with:
  ```python
data = json.dumps({'ids': my_list})
```

- R
  ```r
data <- toJSON(list(ids=mylist))
```
def fetch_endpoint_POST(server, request, data, content_type='application/json '):
    r = requests.post(server+request,
                      headers={"Content-Type" : content_type},
                      data=data )

    if not r.ok:
        r.raise_for_status()
        sys.exit()

    if content_type == 'application/json':
        return r.json()
    else:
        return r.text
R Helper function

```r
fetch_endpoint_POST <- function(server, request, content_type){
  """
  Fetch an endpoint from the server, allow overriding of default content-type
  """
  r <- POST(paste(server, request, sep = """"), content_type(content_type),
              accept(content_type), body = data)

  stop_for_status(r)

  if (content_type == 'application/json'){
    return (fromJSON(content(r, "text")))
  } else {
    return (content(r, "text"))
  }
}
```
Decoding POST queries

POST endpoints return a dictionary of dictionaries.

```json
{
    "ENSG00000157764": {
        "source": "ensembl_havana",
        "object_type": "Gene",
        ...
    },
    "ENSG00000248378": {
        "source": "havana",
        "object_type": "Gene",
        ...
    }
}
```
Decoding POST queries

- You could use your input list as your keys, or you could move through the dictionary with:
  - Python
    
    ```python
    for key, value in post_query.items():
    ```
  - Perl
    
    ```perl
    foreach my $hash_reference (@{$post_query}) {
        my %hash = %$hash_reference;
    }
    ```
  - R just treats these as dataframes
Exercises 6

1. Fetch the all the transcripts of *ESPN* using the `lookup` function. Fetch the cDNA sequences of all transcripts using a single `POST` request, and print in FASTA format.

2. Get all variants that are located on chromosome 17 between 80348215 and 80348333. Get the variant class, evidence attributes, source and the `most_severe_consequence` for all variants in that region from the variant `POST` endpoint.
Rate limiting

Requests are rate limited to prevent a single user from monopolising the resources.

Response headers show we are allowed 55000 requests over an hour (3600 seconds)
An average 15 requests per second
1 request used and 892 sec (~15 minutes) from reset
Rate limiting

Requests are rate limited to prevent a single user from monopolising the resources.

Retry-After: 40.0
X-RateLimit-Limit: 55000
X-RateLimit-Reset: 892
X-RateLimit-Period: 3600
X-RateLimit-Remaining: 54999

Wait 40 seconds before sending another request or...

429
Exercises 7

The Jupyter notebook contains a script that queries the `ping` endpoint 25 times, printing the count, the HTTP Status Code, and the X-RateLimit-Remaining header each time.

1. Increase the number of loops, do you start to get 429 errors?
2. Can you add in a step to make it wait a few seconds every iteration? Or every 100 iterations?
Feedback

training.ensembl.org/events
Help and Documentation

The REST API release notes:
https://github.com/Ensembl/ensembl-rest/wiki/Change-log

Email us helpdesk@ensembl.org
Ensembl public mailing lists dev@ensembl.org, announce@ensembl.org
Ensembl Acknowledgements

The Entire Ensembl Team

Adam Frankish, Ahamed Imran Abdul Salam, Alexandra Bignell, Ameya Chaubal, Andrea Winterbottom, Andrew Berry, Andrew Parton, Andrey Azov, Andy Yates, Anja Thormann, Anmol Jaywant Hemrom, Anne Lyle, Astrid Gall, Benjamin Moore, Bethany Flint, Brandon Walts, Bruno Contreras-Moreira, Carla Cummins, Carlos Garcia Giron, Claire Davidson, Cristina Guijarro, Dan Sheppard, Daniel Zerbino, David Thybert, Denye Ogeh, Diana Lemos, Elizabeth Lewis, Emily Perry, Fergal Martin, Fiona Cunningham, Gareth Maslen, Gareth Williams, Garth Ilsley, Guy Naamati, Helen Schuilenburg, IF Barnes, Ilias Lavidas, Irina Armean, James Allen, Jamie Allen, Jane Loveland, Jonathan Mudge, Jorge Alvarez-Jarreta, Jose Carlos Marugan, Jose Manuel Gonzalez Martinez, Jyothish Bhai, Kamalkumar Jayantilal Dodiya, Kevin Howe, Kieron Taylor, Kostas Billis, Lahcen Campbell, Leanne Haggerty, Luca Da Rin Fioretto, Magali Ruffier, Manoj Sakthivel, Manuel Carbajo Martinez, Marc Chakiachvili, Marek Szuba, Marie-Marthe Suner, Matthew Hardy, Matthew Russell, Matthieu Barba, Matthieu Muffato, Michael Paulini, Michal Szpak, Mike Kay, Mikkel Christensen, Mira Sycheva, Nick Langridge, Nishadi De Silva, Osagie Izuogu, Paul Davis, Paul Flicek, Reemanand Achuthan, Reham Fatima, Ridwan Amode, Ruth Bennett, Sanjay Boddu, Sarah Donaldson, Sarah Hunt, Shamika Mohanan, Stephen Trevanion, Thibaut Hourlier, Thomas Juettemann, Thomas Maurel, Tiago Grego, Toby Hunt, Tuan Le, Vasi Sitnik

Funding

Co-funded by the European Union
Training materials

• Ensembl training materials are protected by a CC BY license
  http://creativecommons.org/licenses/by/4.0/
• If you wish to re-use these materials, please credit Ensembl for their creation
• If you use Ensembl for your work, please cite our papers
  http://www.ensembl.org/info/about/publications.html