

# Acquisition de données

## API & web scraping



Nicolas CASAJUS

{{ Data scientist FRB-Cesab }}

Mardi 3 décembre 2019

# Les API

# Qu'est-ce qu'une API ?

👉 Interface de Programmation Applicative

# Qu'est-ce qu'une API ?

- Protocole HTTP(S) : Affichage d'une page web



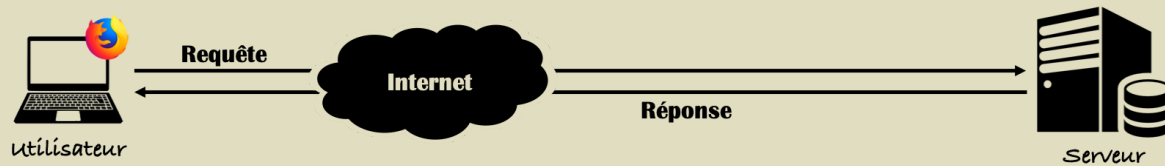
utilisateur



serveur

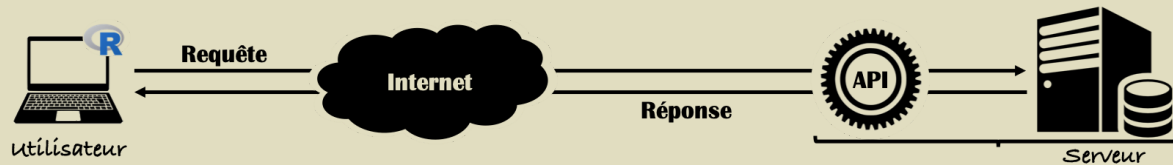
# Qu'est-ce qu'une API ?

- Protocole HTTP(S) : Affichage d'une page web



# Qu'est-ce qu'une API ?

- Requête via API



👉 Accès via interfaces en ligne de commande :  /  /  / 

Automatisation & Reproductibilité

Différents formats de données (xml, (geo)json, yaml, texte, etc.)

# Qu'est-ce qu'une API ?

- Pas de protocole standardisé
- Termes et conditions d'utilisation propres
- Requiert souvent une clé d'authentification (**token**)
- Dans certains cas, quelques limitations (sur-sollicitations, # requêtes, # résultats, etc.)





# Sous de nombreux packages

Package	Type de données
<code>{taxize}</code>	Taxonomie
<code>{spocc}</code>	Occurrences d'espèces
<code>{rcites}</code>	Espèces protégées mondiales
<code>{rfishbase}</code>	Informations sur les poissons
<code>{rredlist}</code>	Accès à IUCN liste rouge
<code>{treebase}</code>	Données phylogénétiques
<code>{traits}</code>	Traits d'espèces
<code>{rnaturalearth}</code>	Données vectorielles spatiales
<code>{raster}</code>	Données climatiques, altitude, etc.
...	...

 Pour aller plus loin : [\[ rOpenSci \]](#) [\[ Open Data \]](#)



# Zoom sur le package `{taxize}`

- *A taxonomic toolbelt for R*
- Permet notamment :
  - vérifier les noms d'espèces
  - obtenir les synonymes
  - obtenir les noms communs ( /  /  / )
  - obtenir la classification taxonomique
  - ...
- Couvre de nombreuses bases de données taxonomiques :
  - Encyclopedia of Life (EOL)
  - Integrated Taxonomic Information Service (ITIS)
  - National Center for Biotechnology Information (NCBI)
  - IUCN Red List
  - World Register of Marine Species (WoRMS)
  - Global Names Resolver
  - ...

👉 L'accès à certaines bases de données nécessite un token

## Zoom sur le package `{taxize}`



*Acanthurus lineatus*

# Zoom sur le package {taxize}

? Le nom d'espèce est-il bien orthographié ?

```
species_name <- "Acanthuurus lineatus"
```

```
taxize::gnr_resolve(names = species_name)
```

##	submitted_name	matched_name	data_source_title	score
## 1	Acanthuurus lineatus	Acanthurus lineatus	NCBI	0.75
## 2	Acanthuurus lineatus	Acanthurus lineatus	Freebase	0.75
## 3	Acanthuurus lineatus	Acanthurus lineatus	Encyclopedia of Life	0.75
## 4	Acanthuurus lineatus	Acanthurus lineatus	iNaturalist	0.75
## ...				

👍 Bonne orthographe : *Acanthurus lineatus*

# Zoom sur le package {taxize}

? Quel est le nom accepté ?

```
species_name <- "Acanthurus lineatus"
```

```
taxize::tnrs(query = species_name)
```

```
##           submittedname      acceptedname sourceid score      matchedname
## 1   Acanthurus lineatus Acanthurus lineatus   NCBI     1 Acanthurus lineatus
```

```
taxize::tnrs(query = species_name)$acceptedname
```

```
## [1] "Acanthurus lineatus"
```

# Zoom sur le package {taxize}

? Quelle est sa classification taxonomique ?

```
taxize::classification(x = species_name, db = "itis")
```

- Total: 1
- Found: 1
- Not Found: 0

```
$`Acanthurus lineatus`
```

	name	rank	id
1	Animalia	kingdom	202423
2	Bilateria	subkingdom	914154
3	Deuterostomia	infrakingdom	914156
4	Chordata	phylum	158852
5	Vertebrata	subphylum	331030
6	Gnathostomata	infraphylum	914179
7	Actinopterygii	superclass	161061
8	Teleostei	class	161105
9	Acanthopterygii	superorder	166082
10	Perciformes	order	167640
11	Acanthuroidei	suborder	172249
12	Acanthuridae	family	172250
13	Acanthurus	genus	172251
14	Acanthurus lineatus	species	172271

# Zoom sur le package {taxize}

? Quels sont ses synonymes ?

```
synonyms <- taxize::synonyms(species_name, db = "worms")
```

✓ Found: Acanthurus lineatus

== Results ==

- Total: 1
- Found: 1
- Not Found: 0

```
synonyms <- as.data.frame(synonyms[[species_name]])  
synonyms[, c("scientificname", "status", "valid_name", "order")]
```

##	scientificname	status	valid_name	order
## 1	Acanthurus vittatus	unaccepted	Acanthurus lineatus	Perciformes
## 2	Chaetodon lineatus	unaccepted	Acanthurus lineatus	Perciformes
## 3	Ctenodon lineatus	unaccepted	Acanthurus lineatus	Perciformes
## 4	Harpurus lineatus	unaccepted	Acanthurus lineatus	Perciformes
## 5	Hepatus lineatus	unaccepted	Acanthurus lineatus	Perciformes
## 6	Rhombotides lineatus	unaccepted	Acanthurus lineatus	Perciformes
## 7	Teuthis lineatus	unaccepted	Acanthurus lineatus	Perciformes

# Zoom sur le package `{taxize}`

❓ Quels sont ses noms communs ?

```
taxize::sci2comm(species_name, db = "ncbi")
```

```
## [1] "lined surgeonfish"
```

```
taxize::sci2comm(species_name, db = "itis")
```

```
## [1] "lined surgeonfish"
```

```
taxize::sci2comm(species_name, db = "worms")
```


```
## [1] "lined surgeonfish" "ニジハギ"
```

👍 En 🇫🇷 : le **Poisson chirurgien rayé** ou **Poisson chirurgien clown**

# Construction d'un client R

*Que faire si aucun package R n'a été développé mais qu'il existe une API ?*

👉 Construire soi-même un client R (c.-à-d. écrire soi-même les requêtes à envoyer à l'API)

Sous  il existe de nombreux packages pour communiquer avec les services Web (`{httr}`, `{curl}`, etc.)

👉 **Getting started with httr**

**Best practices for API packages**

**Managing secrets**

Voyons deux exemples illustrant l'utilisation de `{httr}`



# IUCN Red list API



👉 Conditions d'utilisation disponibles **ici**

Cette API permet d'obtenir différentes informations sur les espèces en danger à l'échelle globale

- Listes d'espèces par catégorie IUCN
- Listes d'espèces par pays
- Les habitats, menaces pour chaque espèce
- *et bien d'autres...*

Elle requière un **token**

# IUCN Red list API

Pour la démonstration, nous utiliserons le token fourni dans la documentation de l'API



👍 Si vous voulez utiliser cette API, **SVP** obtenez votre propre **token**

# IUCN Red list API

- Stockons le token comme variable d'environnement dans le fichier `.Renviron`

```
usethis::edit_r_environ() # Ouverture du fichier `.Renviron`
```

- Et ajoutons cette ligne (en adaptant la valeur)

```
IUCN_KEY=9bb4facb6...
```

- Vérifions que le PAT est bien stocké (après un redémarrage de )

```
Sys.getenv("IUCN_KEY")
```

```
## [1] "9bb4facb6..."
```

# IUCN Red list API

? Combien d'espèces de chaque catégorie IUCN y a-t-il en France ?

- Ecrivons la requête

```
api_url    <- "https://apiv3.iucnredlist.org/api/v3/"
query      <- "country/getspecies/"
country    <- "FR"
iucn_token <- Sys.getenv("IUCN_KEY")

request <- paste0(
  api_url,
  query,
  country,
  "?token=",
  iucn_token
)
```

```
## [1] "https://apiv3.iucnredlist.org/api/v3/country/getspecies/FR?token=9bb4fac..."
```

# IUCN Red list API

- Envoyons la requête à l'API et récupérons le résultat

```
response <- httr::GET(request)
```

- Affichons le statut de la réponse

```
httr::http_status(response)
```

```
## $category  
## [1] "Success"  
##  
## $reason  
## [1] "OK"  
##  
## $message  
## [1] "Success: (200) OK"
```

- Sous quel format sont retournées les données ?

```
httr::http_type(response)
```

```
## [1] "application/json"
```

# IUCN Red list API

- Accédons au contenu de la réponse (données)

```
datas <- httr::content(response, as = "text")
```

```
{
  "count":3897,
  "country":"FR",
  "result":[
    {
      "taxonid":190498,
      "scientific_name":"Abida ateni",
      "subspecies":null,
      "subpopulation":null,
      "category":"VU"
    },
    {
      "taxonid":156905,
      "scientific_name":"Abida attenuata",
      "subspecies":null,
      "subpopulation":null,
      "category":"LC"
    }
    ...
  ]
}
```

# IUCN Red list API

- Convertissons (*parse*) ce format JSON en objet R (package `{jsonlite}`)

```
results <- jsonlite::fromJSON(datas)
```

- Quel est le format retourné ?

```
str(results)
```

```
## List of 3
## $ count : int 3897
## $ country: chr "FR"
## $ result :'data.frame':  3897 obs. of  6 variables:
##   ..$ taxonid      : int [1:3897] 190498 156905 156761 156390 156989 ...
##   ..$ scientific_name: chr [1:3897] "Abida ateni" "Abida attenuata" ...
##   ..$ subspecies    : chr [1:3897] NA NA NA NA ...
##   ..$ rank          : chr [1:3897] NA NA NA NA ...
##   ..$ subpopulation : chr [1:3897] NA NA NA NA ...
##   ..$ category      : chr [1:3897] "VU" "LC" "LC" "LC" ...
```

# IUCN Red list API

- Extrayons les données qui nous intéressent

```
species_fr <- results$result
head(species_fr)
```

```
##   taxonid      scientific_name subspecies rank subpopulation category
## 1  190498      Abida ateni      <NA> <NA>      <NA>      VU
## 2  156905      Abida attenuata  <NA> <NA>      <NA>      LC
## 3  156761      Abida bigerrensis <NA> <NA>      <NA>      LC
## 4  156390      Abida cylindrica <NA> <NA>      <NA>      LC
## 5  156989      Abida gittenbergeri <NA> <NA>      <NA>      NT
## 6  156823      Abida occidentalis <NA> <NA>      <NA>      LC
```

- Nombre d'espèces par catégorie listées par l'IUCN

```
table(species_fr[, "category"])
```

```
##
##   CR   DD   EN   EW   EX   LC LR/cd LR/lc LR/nt   NT   VU
##   44  346  103   1   9  3020   1   3   16  222  266
```

 Package R {rredlist}



# OpenStreetMap Nominatim



👉 Conditions d'utilisation disponibles [ici](#)

Cette API permet d'obtenir les coordonnées géographiques de lieux (villes, bâtiment, numéros de rue, etc.)

Elle ne requière aucun token. Cependant, le nombre de requêtes est limitée à 1/s

👉 `Sys.sleep()`

# OpenStreetMap Nominatim

- Ecrivons le requête de base

<http://nominatim.openstreetmap.org/search/{{@}}?format=json&addressdetails=0&limit=1>

```
api_url <- "http://nominatim.openstreetmap.org/search/"

params <- list(
  format = "json",           # Format des données retournées
  details = 0,               # Recommandé par l'API
  limit = 1                  # Un seul résultat par requête
)
```

- Pour quelle entité géographique ({{@}}) ?

```
location <- "5 rue de l'école de médecine, Montpellier, France"
location <- gsub("\\s+", "%20", location)
```

```
## [1] "5%20rue%20de%20l'école%20de%20médecine,%20Montpellier,%20France"
```

# OpenStreetMap Nominatim

- Ecrivons la requête complète

```
request <- paste0(api_url, location)
```

```
## "http://nominatim.openstreetmap.org/search/5%20rue%20de%20l'école%20de%20méd..."
```

- Envoyons la requête à l'API (avec paramètres) et récupérons le résultat

```
response <- httr::GET(request, query = params)
```

- Affichons le statut de la réponse

```
httr::http_status(response)
```

```
## $category  
## [1] "Success"  
##  
## $reason  
## [1] "OK"  
##  
## $message  
## [1] "Success: (200) OK"
```

# OpenStreetMap Nominatim

- Sous quel format sont retournées les données ?

```
httr::http_type(response)
```

```
## [1] "application/json"
```

- Accédons au contenu de la réponse (données)

```
datas <- httr::content(response, as = "text")
```

```
[
  {
    "place_id":259115419,
    "licence":"Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright",
    "osm_type":"node",
    "osm_id":6405038665,
    "boundingbox":["43.6127241","43.6128241","3.8734359","3.8735359"],
    "lat":"43.6127741",
    "lon":"3.8734859",
    "display_name":"5, Rue de l'École de Médecine, Centre Historique, Montpellier, Hérault...",
    "class":"place",
    "type":"house",
    "importance":0.831
  }
]
```

# OpenStreetMap Nominatim

- Convertissons (parse) ce format JSON en objet R

```
results <- jsonlite::fromJSON(datas)
```

- Quel est le format retourné ?

```
class(results)
```

```
## [1] "data.frame"
```

- Extrayons les données qui nous intéressent

```
results$lon <- as.numeric(results$lon)  
results$lat <- as.numeric(results$lat)  
(xy <- results[, c("lon", "lat")])
```

```
##      lon      lat  
## 1 3.873486 43.61277
```

# OpenStreetMap Nominatim

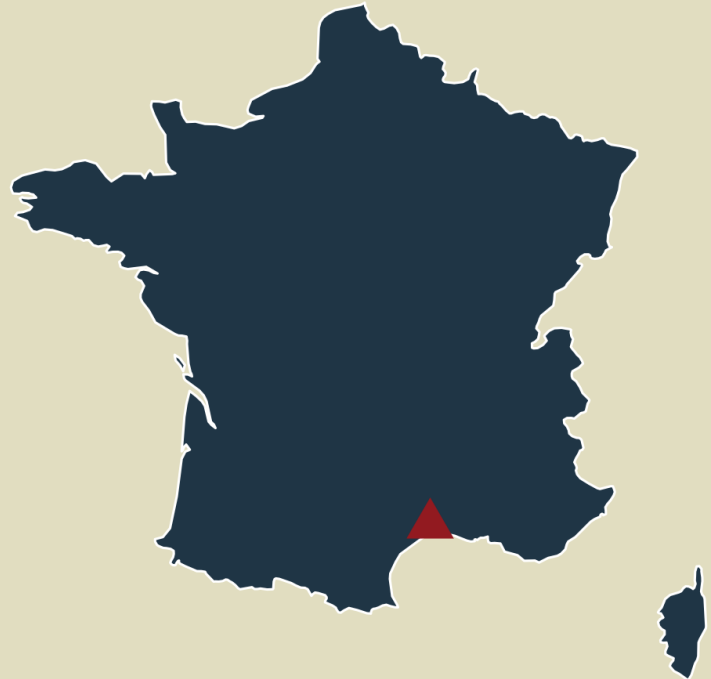
- Fiabilité du résultat ?

```
## Fond de carte -----
```

```
maps::map(  
  region = "France",  
  fill   = TRUE,  
  col    = "#294557",  
  border = "white"  
)
```

```
## Ajout du point -----
```

```
points(  
  xy,  
  pch = 17,  
  col = "#a52a2a",  
  cex = 2  
)
```



# Web scraping

# Quésaco ?

- Extraction via des programmes informatiques (Robots) du contenu de pages Web en vue de son utilisation
- La question de la légalité : le cas Google (le gros foutage de gueule)
- Est-ce que tout peut-être scrapé ?
  - Compte utilisateur
  - Captchas
  - Bannissement d'IP

👍 **Be polite!** with `{polite}`

Sous , deux approches :

1. L'approche brutale (`readLines()` + `regular expressions`)
2. Le package R : `{rvest}`



# Extraction de texte

👉 Source : **Wikipédia**

- Construisons l'URL de la page

```
base_url <- "https://fr.wikipedia.org/wiki/"  
article  <- "Acanthurus_lineatus"  
  
request  <- paste0(base_url, article)
```

- Ouverture d'une session HTML

```
html <- rvest::html_session(request)
```

# Extraction de texte

- Extrayons le titre de la page

```
html %>%  
  rvest::html_nodes(css = ".firstHeading") %>%  
  rvest::html_text()
```

```
## [1] "Acanthurus lineatus"
```

# Téléchargement d'une image

- Extrayons l'URL de la troisième image sur la même page

```
img_url <- html %>%  
  rvest::html_nodes(xpath = '//*/img') %>%  
  rvest::html_attr("src") %>%  
  .[3]
```

```
## [1] "//upload.wikimedia.org/wikipedia/commons/thumb/5/59/Acanthurus_lineatus_2..."
```

- Téléchargeons l'image

```
download.file(  
  url      = paste0("https:", img_url),  
  destfile = "acanthurus_lineatus.jpg"  
)
```

```
essai de l'URL 'https://upload.wikimedia.org/wikipedia/commons/thumb/5/59/Acanthu...'
Content type 'image/jpeg' length 24891 bytes (24 KB)
```

```
=====
downloaded 24 KB
```

# Extraction d'un tableau html

👉 Obtenir le nombre d'habitants des 10 communes les plus peuplées de France (source **Wikipédia**)

- Construisons l'URL de la page

```
base_url <- "https://fr.wikipedia.org/wiki/"  
article  <- "Liste_des_communes_de_France_les_plus_peuplées"  
request  <- paste0(base_url, article)
```

- Ouverture d'une session HTML

```
html <- rvest::html_session(request)
```

# Extraction d'un tableau html

- Extrayons tous les tableaux HTML de la page

```
tables <- rvest::html_table(html, fill = TRUE)
```

- Combien en a-t-on ?

```
length(tables)
```

```
## [1] 4
```

- Seul le premier nous intéresse avec les colonnes **Commune** et **2017**

```
cities <- tables[[1]]  
cities <- cities[-1, c(3, 7)]  
cities <- cities[1:10, ]  
rownames(cities) <- NULL
```

# Extraction d'un tableau html

- Affichons les données

```
cities
```

```
##      Commune Population légale
## 1      Paris      2 187 526
## 2  Marseille      863 310
## 3      Lyon      516 092
## 4  Toulouse      479 553
## 5      Nice      340 017
## 6      Nantes      309 346
## 7  Montpellier      285 121
## 8  Strasbourg      280 966
## 9      Bordeaux      254 436
## 10     Lille      232 787
```

👉 Un peu de nettoyage est nécessaire

# Extraction d'un tableau html

- Renommons les colonnes

```
colnames(cities) <- c("city", "pop2017")
```

- Nettoyons le nombre d'habitants

```
cities$pop2017 <- gsub("[[:punct:]]|[[:space:]]", "", cities$pop2017)  
cities$pop2017 <- as.numeric(cities$pop2017)
```

```
##           city pop2017  
## 1      Paris 2187526  
## 2  Marseille 863310  
## 3       Lyon 516092  
## 4   Toulouse 479553  
## 5       Nice 340017  
## 6    Nantes 309346  
## 7 Montpellier 285121  
## 8   Strasbourg 280966  
## 9    Bordeaux 254436  
## 10      Lille 232787
```

# Bonus : Une bubble map

```
api_url <- "http://nominatim.openstreetmap.org/search/"
params <- list(format = "json", details = 0, limit = 1)

(locations <- paste0(cities$city, ",%20France"))
```

```
## [1] "Paris,%20France"      "Marseille,%20France"  "Lyon,%20France"
## [4] "Toulouse,%20France"   "Nice,%20France"       "Nantes,%20France"
## [7] "Montpellier,%20France" "Strasbourg,%20France" "Bordeaux,%20France"
## [10] "Lille,%20France"
```

```
xy <- data.frame()

for (location in locations) {

  request <- paste0(api_url, location)
  response <- httr::GET(request, query = params)

  datas <- httr::content(response, as = "text")
  results <- jsonlite::fromJSON(datas)

  tmp <- results[, c("lon", "lat")]
  xy <- rbind(xy, tmp)

  Sys.sleep(1)
}
```



# Bonus : Une bubble map

```
(xy <- cbind(cities, xy))
```

##	city	pop2017	lon	lat
## 1	Paris	2187526	2.3514992	48.85661
## 2	Marseille	863310	5.3699525	43.29617
## 3	Lyon	516092	4.8320114	45.75781
## 4	Toulouse	479553	1.4442469	43.60446
## 5	Nice	340017	7.2683912	43.70094
## 6	Nantes	309346	-1.5541362	47.21864
## 7	Montpellier	285121	3.8767337	43.61124
## 8	Strasbourg	280966	7.7507127	48.58461
## 9	Bordeaux	254436	-0.5800364	44.84123
## 10	Lille	232787	3.0635282	50.63657

```

## Fond de carte      -----
maps::map(
  region = "France",
  fill   = TRUE,
  col    = "#294557",
  border = "white"
)

## Ajout des points   -----
points(
  x  = xy$lon,
  y  = xy$lat,
  pch = 19,
  col = "#a52a2a88",
  cex = sqrt(cities$pop2017)/(100*pi)
)

## Ajout des villes   -----
text(
  x      = xy$lon,
  y      = xy$lat,
  labels = xy$city,
  col    = "white",
  pos    = 3,
  cex    = 0.75
)

```



# Pratique