

Modularité des programmes

Introduction :

Ce cours porte sur la modularité en développement logiciel. Le découpage des programmes en composants ou modules logiciels favorise leur maintenance et leur réutilisation dans différents contextes.

Ces modules ou bibliothèques permettent d'enrichir les fonctionnalités d'un langage de programmation selon les besoins. Nous présenterons dans un premier temps l'utilisation et la création de modules au niveau local. Puis, nous nous intéresserons ensuite à l'accès à des modules à distance sur le web par l'intermédiaire d'interfaces applicatives de programmation appelées « API ».

1 | Bibliothèques logicielles

Dans cette première partie, nous aborderons l'utilisation de modules logiciels préexistants ; puis dans un second temps nous étudierons la création d'un nouveau module.

Les langages informatiques disposent d'un certain nombre de fonctionnalités natives. Ainsi le langage Python nous permet d'effectuer des calculs usuels et de traiter plusieurs types de données. Ces fonctionnalités natives permettent de répondre aux besoins courants et récurrents des développeur·se·s.

a. Intérêt des bibliothèques logicielles

Les **fonctionnalités natives** des langages peuvent être étendues en cas de besoin par le recours à des modules complémentaires, appelés de manière générique **bibliothèques logicielles**.



En langage informatique, une bibliothèque est un ensemble de modules regroupés et mis à disposition, afin de ne pas les réécrire à chaque fois.

Ces modules spécialisés permettent d'enrichir le langage en fournissant des outils logiciels dédiés à toutes sortes de problématiques : calcul mathématique, accès au système d'exploitation, cryptographie, communication réseau, protocoles Internet, pour ne citer que quelques familles de bibliothèques.



La bibliothèque standard de Python comporte des centaines de composants logiciels. Leur longue liste est consultable à [cette adresse](#).

Cette riche bibliothèque standard peut, en outre, être complétée si nécessaire par le recours à des modules supplémentaires externes, comme nous allons le voir à présent.

Utilisation d'une bibliothèque

Afin d'illustrer de manière concrète l'utilisation d'une bibliothèque, nous choisissons comme exemple le module externe nommé « Requests ». C'est une bibliothèque HTTP très utilisée et très appréciée par les développeur·se·s Python. Elle ne fait pas partie de la bibliothèque standard, mais elle est officiellement recommandée dans la documentation Python comme interface client HTTP de plus haut niveau.

Son installation, simple et rapide, est présentée dans la documentation en ligne de Requests consultable à [cette adresse](#).



Un·e développeur·se doit savoir se documenter et avoir le réflexe de consulter les différentes documentations proposées. Ces dernières permettent de comprendre et d'utiliser la bibliothèque en question.

Les documentations en ligne des langages et modules sont toujours disponibles en anglais. Certaines sont traduites dans d'autres langues, mais

la traduction n'est pas systématique.



Dans le cas de la bibliothèque Requests, l'aide en ligne a fait l'objet d'une traduction en français, consultable à [cette adresse](#).

Une fois le module installé, il suffit de l'importer pour accéder aux fonctionnalités qu'il propose, ainsi qu'à une aide intégrée locale, donc accessible en l'absence de connexion Internet. Rédigée en langue anglaise, cette aide destinée à un public international est généralement formulée de manière assez simple.

Depuis Python, on importe le module Requests : `import requests`

Nous pouvons interroger le système d'aide natif de Python avec `help()` en précisant le nom du module qui nous intéresse.

```
help(requests)
```

Affiche le texte d'aide suivant :

```
Help on package requests:
```

```
NAME
```

```
requests
```

```
DESCRIPTION
```

```
Requests HTTP Library
```

```
~~~~~
```

```
Requests is an HTTP library, written in Python, for human beings. Basic GET
```

```
usage:
```

```
>>> import requests
```

```
>>> r = requests.get('https://www.python.org')
```

```
>>> r.status_code
```

```
200
```

```
>>> 'Python is a programming language' in r.content
```

```
True
```

... or POST:

```
>>> payload = dict(key1='value1', key2='value2')
```

```
>>> r = requests.post('https://httpbin.org/post', data=payload)
```

```
>>> print(r.text)
```

```
{  
  ...  
  "form": {  
    "key2": "value2",  
    "key1": "value1"  
  },  
  ...  
}
```

The other HTTP methods are supported - see `requests.api`.

Full documentation

is at [<http://python-requests.org.>](http://python-requests.org)

:copyright: (c) 2017 by Kenneth Reitz.

:license: Apache 2.0, see LICENSE for more details.

PACKAGE CONTENTS

`__version__`

`_internal_utils`

`adapters`

`api`

`auth`

`certs`

`compat`

`cookies`

`exceptions`

`help`

`hooks`

`models`

`packages`

`sessions`

status_codes

structures

utils

FUNCTIONS

`checkcompatibility(urllib3version, chardet_version)`

DATA

`__author_email__ = 'me@kennethreitz.org'`

`__build__ = 139520`

`__cake__ = '\n *\n *'\n`

`__copyright__ = 'Copyright 2018 Kenneth Reitz'`

`__description__ = 'Python HTTP for Humans.'`

`__license__ = 'Apache 2.0'`

`__title__ = 'requests'`

`__url__ = 'http://python-requests.org'`

`codes = <lookup 'status_codes'>`

`cryptography_version = '2.6.1'`

VERSION

`2.21.0`

AUTHOR

`Kenneth Reitz`

On observe que cette aide intégrée propose deux exemples de code illustrant des fonctionnalités du module. Ces exemples sont construits avec les deux requêtes HTTP les plus courantes : GET et POST.

Consultons l'aide applicable à GET.

`help(requests.get)`

Nous obtenons le texte suivant :

Help on function get in module requests.api:

`get(url, params=None, **kwargs)`

Sends a GET request.

:param url: URL for the new :class:`Request` object.
:param params: (optional) Dictionary, list of tuples or bytes to send
in the body of the :class:`Request`.
:param **kwargs: Optional arguments that ``request`` takes.
:return: :class:`Response` object
:rtype: requests.Response

L'aide contextuelle fournie par les **IDE** (pour *Integrated Development Environment*, soit « environnement de développements ») facilite également le travail des développeur·se·s en apportant les éléments de syntaxe dans l'éditeur de code, en regard immédiat de la ligne de code courante.

Voici à quoi ressemble le texte d'aide contextuelle associé à la fonction `get()` de Requests :

```
[ ]: reponse = requests.get()
```

Signature: requests.get(url, params=None, **kwargs)
Docstring:
Sends a GET request.

:param url: URL for the new :class:`Request` object.
:param params: (optional) Dictionary, list of tuples or bytes to send
in the body of the :class:`Request`.
:param **kwargs: Optional arguments that ``request`` takes.
:return: :class:`Response` object
:rtype: requests.Response
File: /anaconda3/lib/python3.7/site-packages/requests/api.py
Type: function

La consultation de la documentation en ligne et de l'aide contextuelle nous guide pour l'utilisation de la bibliothèque. Sa lecture nous apprend que nous pouvons facilement requérir des ressources du web à partir de leur URL.

Effectuons une requête HTTP de type GET sur l'URL correspondant à la page Wikipédia consacrée au langage Python en langue française.

```
reponse =  
requests.get('https://fr.wikipedia.org/wiki/Python_(langage)')
```

La fonction `dir()` appelée sur l'objet `reponse` correspondant au résultat de notre requête GET, nous permet de connaître ses attributs.

dir(reponse)

Nous obtenons l'affichage suivant :

['__attrs__',
 '__bool__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__enter__',
 '__eq__',
 '__exit__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__nonzero__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',

 '__setstate__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',

'_content',
'_content_consumed',
'_next',
'apparent_encoding',
'close',
'connection',
'content',
'cookies',
'elapsed',
'encoding',
'headers',
'history',
'ispermanentredirect',
'is_redirect',
'iter_content',
'iter_lines',
'json',
'links',
'next',
'ok',
'raise_for_status',
'raw',
'reason',
'request',
'status_code',
'text',
'url'],

Sans tenir compte des nombreuses méthodes spéciales (celles commençant par les caractères « `_` ») nous constatons l'existence d'une vingtaine d'attributs rattachés à notre objet `reponse`.

L'examen de ces attributs montre qu'il est possible d'accéder facilement à toutes sortes d'informations liées à HTTP, entre autres :

- le code statut retourné par le serveur avec la méthode `status_code` ;
- les entêtes avec la méthode `headers` ;

- et le code source html avec la méthode `text`.

Nous pouvons aussi vérifier avec la méthode `is_redirect` si une redirection a eu lieu et contrôler l'url finale avec la méthode `url`.

```
print(reponse.is_redirect)
```

```
# affiche False
```

```
print(reponse.status_code)
```

```
# affiche 200
```



À retenir

Les fonctions `help()` et `dir()` permettent de se documenter sur le contenu et le fonctionnement des bibliothèques.

c. Création d'un module

Les développeur·se·s peuvent non seulement utiliser les modules existants, mais aussi **créer leurs propres modules logiciels**.

Nous allons l'illustrer avec un exemple simple, en créant un module dédié aux conversions de températures entre les degrés Celsius et Fahrenheit. Notre module pourra également indiquer s'il gèle pour toute température exprimée (en Celsius, comme en Fahrenheit).

Nous définissons notre module avec le code ci-après, que nous enregistrons dans un fichier nommé `temperatures.py`.

```
"""
```

```
Module de conversion de températures Celsius et Fahrenheit.
```

```
Ce module permet :
```

```
- d'effectuer des conversions de températures entre des degrés Celsius et des degrés Fahrenheit et réciproquement
```

```
- d'indiquer si une température, exprimée en Celsius ou en Fahrenheit, est une température de gelée.
```

```
"""
```

```
def celsius_en_fahrenheit(degres_celsius):
```

```
    """Calcule la température en degrés Fahrenheit
```

```

    équivalente à la température en degrés Celsius fournie en argument
    selon la formule température en Fahrenheit = température en Celsius x 9/5 + 32.
    """
    return degres_celsius * 9 / 5 + 32

```

```

def fahrenheit_en_celsius(degres_fahrenheit):
    """Calcule la température en degrés Celsius
    équivalente à la température en degrés Fahrenheit fournie en argument
    selon la formule température en Celsius = (température en Fahrenheit - 32) x 5/9.
    """
    return (degres_fahrenheit - 32) * 5 / 9

```

```

def gel(temperature, unite='C'):
    """Indique s'il gèle en fonction de la température fournie.
    Les degrés sont exprimés soit en degrés Celsius (C, choix par défaut), soit en degrés
    Fahrenheit (F).
    La fonction retourne :
    - la valeur booléenne True si la température est inférieure ou égale à 0 °C
    - la valeur booléenne False si la température est supérieure à 0 °C. """
    if unite.upper() not in ['C', 'F']:
        raise ValueError("l'unité de température doit être en degrés C (pour Celsius) ou F pour
        Fahrenheit")
    if unite == 'F':
        temperature = fahrenheit_en_celsius(temperature)
    return temperature <= 0

```

Nous pouvons ensuite importer ce module dans nos programmes, et ainsi accéder à l'ensemble des fonctionnalités qu'il propose.



Sur la console, passer les commandes suivantes :

```

import sys
sys.path.insert(index,r."/path/to/your/packageOrmodule")

```

où index indique la position du dossier où se trouve le module dans la liste des chemins contenue dans `sys.path`.

On notera que ce code de notre module comporte plusieurs textes de documentation (appelés ***docstrings*** en anglais) matérialisés par les textes multi-lignes encadrés entre triples guillemets : « `"""` ».

Ils sont utiles non seulement pour la maintenance du code, mais pas seulement, comme nous allons le mettre en évidence dans un instant.

Une fois notre module réalisé, nous pouvons en importer et en utiliser les fonctions en quelques lignes de code.

```
import temperatures

print(temperatures.celsius↵fahrenheit(0))
# affiche 32.0
print(temperatures.fahrenheit↵celsius(212))
# affiche 100.0
print(temperatures.gel(0))
# affiche True
print(temperatures.gel(33, 'F'))
# affiche False
print(temperatures.gel(32, 'F'))
# affiche True
```

Le fait d'avoir renseigné notre module et ses fonctions avec des *docstrings* permet à Python de proposer l'aide correspondante avec les fonctions natives de l'aide système.

```
help(temperatures)
```

Nous obtenons l'affichage suivant :

```
Help on module temperatures:

NAME
    temperatures - Module de conversion de températures Celsius et Fahrenheit.

DESCRIPTION
```

Ce module permet :

- d'effectuer des conversions de températures entre des degrés Celsius et des degrés Fahrenheit et réciproquement
- d'indiquer si une température, exprimée en Celsius ou en Fahrenheit, est une température de gelée.

FUNCTIONS

celsius_en_fahrenheit(degrees_celsius)

Calcule la température en degrés Fahrenheit

équivalente à la température en degrés Celsius fournie en argument

selon la formule température en Fahrenheit = température en Celsius x 9/5 + 32.

fahrenheit_en_celsius(degrees_fahrenheit)

Calcule la température en degrés Celsius

équivalente à la température en degrés Fahrenheit fournie en argument

selon la formule température en Celsius = (température en Fahrenheit - 32) x 5/9.

gel(temperature, unite='C')

Indique s'il gèle en fonction de la température fournie.

Les degrés sont exprimés soit en degrés Celsius (C, choix par défaut), soit en degrés Fahrenheit (F).

La fonction retourne :

- la valeur booléenne True si la température est inférieure ou égale à 0 °C

- la valeur booléenne False si la température est supérieure à 0 °C.

temperatures

Module de conversion de températures Celsius et Fahrenheit.

Ce module permet :

- d'effectuer des conversions de températures entre des degrés Celsius et des degrés Fahrenheit et réciproquement
- d'indiquer si une température, exprimée en Celsius ou en Fahrenheit, est une température de gelée.

Functions

celsius_en_fahrenheit(degrees_celsius)

Calcule la température en degrés Fahrenheit

équivalente à la température en degrés Celsius fournie en argument

selon la formule température en Fahrenheit = température en Celsius x 9/5 + 32.

fahrenheit_en_celsius(degrees_fahrenheit)

Calcule la température en degrés Celsius

équivalente à la température en degrés Fahrenheit fournie en argument

selon la formule température en Celsius = (température en Fahrenheit - 32) x 5/9.

gel(temperature, unite='C')

Indique s'il gèle en fonction de la température fournie.

Les degrés sont exprimés soit en degrés Celsius (C, choix par défaut), soit en degrés Fahrenheit (F).

La fonction retourne :

- la valeur booléenne True si la température est inférieure ou égale à 0 °C

- la valeur booléenne False si la température est supérieure à 0 °C.

La *docstring* figurant au début du module est prise en compte de la manière suivante :

- la première ligne, associée au nom du module, en fournit une description synthétique dans la section « NAME » (nom) ;
- les lignes suivantes apparaissent dans la section « DESCRIPTION » ;
- les *docstrings* des fonctions sont présentées individuellement dans la section « FUNCTIONS » (fonctions) à la suite de leur nom et de leurs paramètres.



Notre module exemple a été développé avec un paradigme fonctionnel mais la modularité n'impose aucun paradigme en particulier. Nous pourrions tout aussi bien développer ce module avec un paradigme objet. Le système d'aide native présenterait alors les attributs et méthodes des classes définies dans le module.

Nous savons maintenant utiliser et créer des modules au niveau local. Découvrons maintenant ce qu'il est possible de réaliser avec une connexion au web.

2 | Accès aux API web

Intéressons-nous aux possibilités d'interactions avec des modules distants par le biais d'API web. Commençons par définir et caractériser ce qu'est une API web.

a. Définition



API :

Une API (de l'anglais *Application Programming Interface*) est une interface de programmation applicative (ou interface de programmation

d'application).

Le sigle API désigne de manière générique tout type d'interface applicative. Dans le cadre de ce cours nous nous intéressons aux API du web.

Caractérisation d'une API web



Définition

API web :

On désigne par « API web » une API fournie par un serveur web.



Exemple

Les API web sont nombreuses et fournissent des données très variées. En voici quelques-unes ci-dessous à titre d'exemples.

- Open Weather Map : fournit des informations météo diverses relatives au lieu qui lui est précisé.
- Text-to-speech : convertit le texte qui lui est transmis en fichier audio.
- Géo : cette API web, produite par le gouvernement français, délivre des informations sur une commune (code postal, coordonnées géographiques, regroupements auxquels elle appartient).
- Paypal, HiPay, Stripe... : solutions de paiements en ligne accessibles via des API.

Cette API expose ses données via des **points d'accès** (**endpoints** en anglais) qu'il est possible de consulter par le biais de requêtes HTTP. Les URL des points d'accès sont indiquées dans la documentation de l'API. Les réponses sont généralement proposées au format **JSON** ou parfois en **XML**.

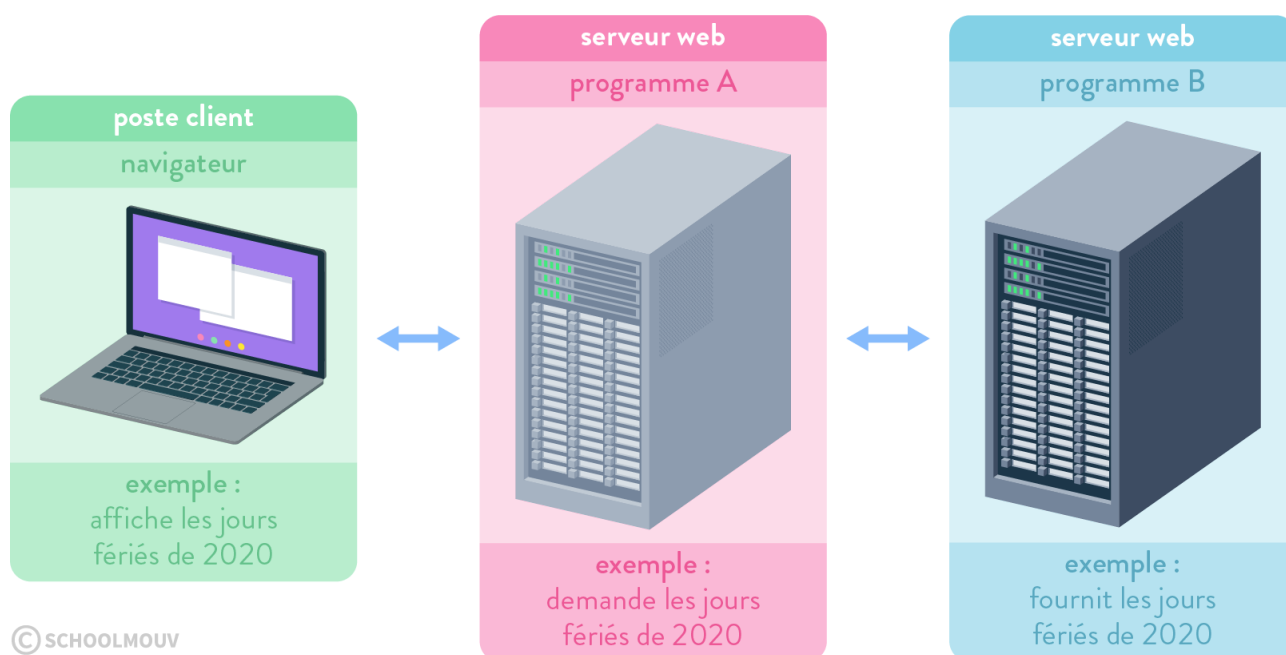
Il existe différentes manières de proposer une API dans le contexte du web, mais ces API respectent généralement le modèle appelé **REST** (pour *Representational State Transfer*).

Ce modèle est basé sur des paires de requêtes et de réponses caractérisées par l'absence de notion d'état entre ces requêtes qui sont totalement indépendantes les unes des autres, en conformité avec la logique du protocole HTTP. Une API respectant le modèle REST est appelée **RESTful**.

Les interactions avec les API web reposent sur l'emploi de verbes HTTP : principalement GET pour consommer des données, éventuellement complété par POST, PUT et DELETE si l'API permet aussi de créer et d'éditer des données sur le serveur.



Des bibliothèques Python spécialisées telles que Bottle, Flask ou Django permettent de proposer des API web et notamment des API respectant le modèle REST.



Modalités d'accès aux API

Les API web possèdent des modalités d'accès variables. Celles-ci sont définies par le producteur du service et peuvent être caractérisées selon plusieurs critères :

- coût (gratuit ou payant) ;


- utilisateur·rices·s (accessible à tou·te·s ou seulement à un public restreint) ;
- mode d'accès (avec ou sans clé d'accès) ;
- utilisation avec ou sans limitation (en général sous la forme d'un nombre de requêtes par unité de temps) ;
- réutilisation, avec ou sans restriction d'exploitation des données, et le cas échéant une distinction selon que la réutilisation soit ou non à vocation commerciale.

Les points d'accès à l'API sont indiqués par la documentation, ainsi que nous allons l'illustrer en interrogeant de manière concrète une API web.



Exemple

Voici un exemple de conditions d'accès à une API : l'API Géo. Celle-ci est ouverte à tous, mais dans des conditions bien précises : **10** appels/seconde/ip. En effet, en sollicitant l'API, on mobilise des ressources informatiques, et celles-ci ne sont pas gratuites !


API Géo
Interrogez les référentiels géographiques plus facilement

Accès

API ouvert à tous

Disponibilité

100.00% actif / dernier mois

Activité

Dernière modification le 17/01/2020

Producteur

Direction interministérielle du numérique

Limite d'usage

10 appels / seconde / IP

Description

Accès

Support

Supervision

Limite d'usage

Partenaires

Documentation technique

Services

Description

L' **API Géo** est une boîte-à-outils **facile à prendre en main** pour rendre vos applications et bases de données plus intelligentes, en terme de positionnement et de connaissance des territoires.

Grâce à elle vous pouvez notamment :

- Rechercher des communes par nom, code postal ou coordonnées géographiques
- Connaître les groupements auxquels appartient une commune, ainsi que leurs compétences (*bientôt*)
- Savoir si une parcelle appartient à certains zonages (appellations d'origine, quartiers prioritaires...) (*bientôt*)

Les différentes fonctionnalités arrivant progressivement, restez informés en suivant cette page ou [en nous contactant](#).

Informations complémentaires

Couverture du territoire

France métropolitaine et [DROM](#).

Coordonnées géographiques

Cette API utilise exclusivement des coordonnées géographiques [WGS-84](#). Elle peut renvoyer des données au format JSON ou [GeoJSON](#).

Qui peut utiliser cette API ?

Tout le monde. Mais si vos besoins sont massifs, [contactez-nous](#) au préalable.

Données source

[Toutes les données utilisées](#) sont sous licences Open Data.

Mail de contact


geo@api.gouv.fr

Accès

L'API est ouverte à tous.

Support

Vous pouvez contacter le support de cette API par mail .

 Envoyer un mail à geo@api.gouv.fr

d.

Interrogation d'une API

L'interrogation de l'API sera effectuée à l'aide de la bibliothèque Requests présentée en première partie de ce cours.



Exemple

L'API que nous utiliserons porte sur les jours fériés en France.

La documentation de l'API est consultable à [cette adresse](#). On y trouve également le code source et la licence d'utilisation.

Cette API ne nécessite pas de clé d'accès pour sa consultation.

Cette documentation nous précise que les points d'accès fournissent des données au format JSON, et nous fournit le format de l'URL à laquelle envoyer notre requête pour connaître tous les jours fériés d'une année :

```
https://jours-feries-france.antoine-augusti.fr/api/:annee
```



« :annee » doit être remplacé par l'année de notre choix.

Effectuons la requête GET correspondante pour l'année 2020.

```
import requests
```

```
reponse = requests.get('https://jours-feries-france.antoine-augusti.fr/api/2020')
```

```
print(reponse.json())
```

Ce code produit l'affichage suivant :

```
[{'date': '2020-01-01', 'nom_jour_ferie': 'Jour de l'an'},  
{ 'date': '2020-04-13', 'nom_jour_ferie': 'Lundi de Pâques'},  
{ 'date': '2020-05-01', 'nom_jour_ferie': 'Fête du travail'},  
{ 'date': '2020-05-08', 'nom_jour_ferie': 'Victoire des alliés'},  
{ 'date': '2020-05-21', 'nom_jour_ferie': 'Ascension'},  
{ 'date': '2020-06-01', 'nom_jour_ferie': 'Lundi de Pentecôte'},  
{ 'date': '2020-07-14', 'nom_jour_ferie': 'Fête Nationale'},  
{ 'date': '2020-08-15', 'nom_jour_ferie': 'Assomption'},  
{ 'date': '2020-11-01', 'nom_jour_ferie': 'Toussaint'},  
{ 'date': '2020-11-11', 'nom_jour_ferie': 'Armistice'},  
{ 'date': '2020-12-25', 'nom_jour_ferie': 'Noël'}]
```

Nous pourrions ensuite facilement extraire et traiter des éléments parmi ces données obtenues via l'API web. L'accès aux données du web sous forme d'API n'est guère plus compliqué que l'accès à des données stockées dans un fichier local.

Les modalités d'interactions varient d'une API à l'autre : certaines sont très simples et proposent uniquement quelques points d'accès en consultation seule, quand d'autres, plus complexes, peuvent comporter de nombreux points d'accès et supporter plusieurs méthodes, afin de non seulement obtenir mais également transmettre ou éditer des données sur le serveur.

Conclusion :

Nous avons abordé la modularité du développement logiciel sous l'angle du découpage de programmes en composants spécialisés ; en montrant d'abord l'utilisation de modules existants, puis la création de tels modules, afin d'enrichir les fonctionnalités natives des langages de programmation.

Nous avons ensuite présenté l'accès web à des modules distants par le biais d'API dont nous avons précisé les caractéristiques techniques et les modalités d'accès, lesquelles varient en fonction des choix opérés par le producteur de la ressource. Nous avons pu constater à l'aide d'un exemple que les API web permettent d'obtenir facilement des données formatées, conçues pour être ensuite traitées par un programme informatique.