Modularité

I. API

Définition 0.1.
Une API (**A**pplication **P**rogramming **I**nterface) permet aux développeur d'intégrer une application à une autre. Cela peut permettre par exemple de récupérer des données structurées depuis un site web pour les exploiter de manière automatisée dans un programme.

1. Exemple

Nous cherchons à récupérer dans un programme python la température et à l'afficher. Pour cela, nous allons utiliser une *API* existante.

Nous allons utiliser les services d'*openweathermap* (faites une recherche internet pour acceder au site). Si vous vous rendez sur le site, vous voyez bien les informations qui nous intéressent mais sous cette forme, nous ne pouvons pas les exploiter facilement. Si vous cliquez sur l'onglet *API*, vous verrez tout ce que le site propose comme API. Certaines sont gratuites d'accès, d'autres payantes Dans tous les cas, pour en bénéficier, il faudra créer un compte sur le site. Pour découvrir le principe de l'utilisation de ces API, nous n'aurons pas besoin de compte car une API de démonstration suffit pour tester notre programme. Libre à vous si vous souhaitez travailler sur des données réelles et actualises de créer un compte.

Nous utiliserons dans cet exemple l'API : https ://openweathermap.org/current.

On accède à une API via une URL fournie par le site. Un exemple d'une telle URL est : https ://samples.openweathermap.org/data/2.5/weather?zip=14200,frappid=439d4b804bc8187953

Cette URL contient plusieurs paramètres :

— zip : code postal et pays
— appid : clé fournie par le site quand vous avez créé un compte. Celle qui est proposée ici donne accès à des informations de démonstration. Elles ont le bon format pour nous permettre de développer et tester notre programme.

Si vous ouvrez cette URL avec FireFox, ce dernier va les présenter de manière lisible. En réalité, le fichier qui est envoyé via cette URL est un fichier au format **json** qui ressemble à *une structure de dictionnaire* avec des association clé-valeur.

```
{
"coord":{"lon":-122.09,"lat":37.39},
"weather":[{"id":500,"main":"Rain","description":"light rain"
   ,"icon":"10d"}],
"base":"stations",
"main":{"temp":280.44,"pressure":1017,"humidity":61,"temp_min
   ":279.15,"temp_max":281.15},
"visibility":12874,
"wind":{"speed":8.2,"deg":340,"gust":11.3},
"clouds":{"all":1},
"dt":1519061700,
"sys":{"type":1,"id":392,"message":0.0027,"country":"US","
   sunrise":1519051894,"sunset":1519091585},
"id":0,
"name":"Mountain View",
"cod":200
}
```

Grâce au module requests, Python peut récupérer les données au format **json** et les intégrer dans un dictionnaire.

Voici un programme basique permettant de récupérer et afficher quelques informations :

```
import requests

API_KEY = "439d4b804bc8187953eb36d2a8c26a02"

def get_weather(api_key, location):
url = f"https://samples.openweathermap.org/data/2.5/weather?
   zip={location}&appid={api_key}"
r = requests.get(url)
return r.json()
```

la fonction get_weather va renvoyer directement un dictionnaire :

```
>>> weather = get_weather(API_KEY, "14000, fr")
>>> print(weather)

{'coord': {'lon': -122.09, 'lat': 37.39},
'weather': [{'id': 500,
        'main': 'Rain',
        'description': 'light rain',
        'icon': '10d'}],
'base': 'stations',
'main': {'temp': 280.44,
        'pressure': 1017,
        'humidity': 61,
        'temp_min': 279.15,
        'temp_max': 281.15},
'visibility': 12874,
'wind': {'speed': 8.2, 'deg': 340, 'gust': 11.3},
'clouds': {'all': 1},
'dt': 1519061700,
'sys': {'type': 1,
        'id': 392,
        'message': 0.0027,
        'country': 'US',
        'sunrise': 1519051894,
        'sunset': 1519091585},
'id': 0,
'name': 'Mountain View',
'cod': 200}
```

Récupérer la température se fait alors via

```
>>> print(weather['main']['temp'])
```

En effet, notre dictionnaire sous la clé ''main' renvoie un nouveau dictionnaire

```
{'temp': 280.44,
        'pressure': 1017,
        'humidity': 61,
        'temp_min': 279.15,
        'temp_max': 281.15}
```

qui à son tour renvoie la température via sa clé 'temp'.

2. Exercice

1. Récupérer de l'exemple précédent la direction du vent (en degrés).
2. En utilisant *Hourly Forecast 4days* (https://openweathermap.org/api/hourly-forecast), afficher les prévisions météo pour les 4 prochains jours.

II. Créer un module en python

Vous l'avez vu, le travail sur les API a été grandement facilité par l'utilisation du module requests qui fait en réalité tout le travail pour nous. Grâce à ce module, notre programme est très concis et lisible. De plus, nous pouvons exploiter ce même module requests dans plusieurs programmes sans avoir à le redévelopper.

Nous allons voir maintenant comment créer nos propres modules en python afin d'enrichir le langage de nouvelles fonctions et objets réutilisables facilement.

1. Principe Général

Un module est un ensemble de fonctions suffisamment générales pour être réutilisables dans plusieurs projets. Pour plus de clarté, on ne regroupera dans un même module que les fonctions relatives à une même fonctionnalité.

Un module s'**importe** par son nom qui est le nom du fichier python privé de l'extension .py.

Par exemple, le module random regroupe des fonctions en lien avec l'aléatoire (randint, choice, shuffle, ...)

Un module étant *une boîte noire* - on a pas en général le code des fonctions sous les yeux - , celui-ci doit exposer de manière claire au développeur son *interface*, c'est à dire

— la liste des fonctions et leur rôle
— ce que chaque fonction prend en entrée
— ce que chaque fonction renvoie en résultat

Ces fonctions sont stockées dans un fichier random.py quelque-part sur le système de fichiers de votre ordinateur.

Pour cela, un formalisme particulier existe sous python - les *docstrings* - qui permet de décrire précisément ces informations. Ainsi, un développeur pourra accéder à l'interface du module pour savoir comment utiliser chacune des fonctions.

2. Exemple

Tapez ce code dans une console python :

```
>>> from random import randint
>>> help(randint)
```

Vous constaterez l'affichage de ce texte explicatif. Cela est rendu possible par l'utilisation de *docstrings* à l'intérieur de la fonction randint.

```
Help on method randint in module random:

randint(a, b) method of random.Random instance
Return random integer in range [a, b], including both end
    points.
```

Si vous tapez help(random) vous aurez une description générale des fonctions disponibles dans le module.

Il est intéressant de regarder les sources d'un module officiel de python pour voir comment sont implémentées toutes ces docstrings.

On retiendra :

— Une *docstring* est une chaîne de caractère délimitée par doubles quotes
— un module commence par une *docstring* de présentation générale
— chaque fonction commence par une *docstring* présentant le rôle et le fonctionnement de celle-ci.

Voici l'implémentation de la fonction randint dans le module random qui montre d'où provient le texte d'aide affiché par la fonction help().

```
def randint(self, a, b):
"""Return random integer in range [a, b], including both end
    points.
"""

return self.randrange(a, b+1)
```

Pour en savoir plus sur les docstrings, vous pouvez vous référer au guide de préconisations python PEP-257 : https ://www.python.org/dev/peps/pep-0257/

3. Importer un Module

Vous êtes confrontés à l'import de modules depuis la première ligne de python que vous avez rencontré pratiquement. En effet, si vous voulez utiliser des fonctions aléatoires ou mathématiques de base, vous êtes obligé d'importer le module random ou maths.

R **ATTENTION ATTENTION : N'importez JAMAIS un module par la commande** from module import *

et pourtant, c'est la première ligne que l'on trouve lorsque l'on programme en python sur calculatrice. C'est **une très mauvaise pratique** menant à des bugs difficiles à déceler. En effet, cela importe toutes les fonctions du module sans distinction, avec le risque d'écraser d'autres fonctions de même nom dans d'autres modules.

Vous trouverez beaucoup de code python sur internet important les modules ainsi. C'est mal, ne le faîtes pas !

Il y a deux moyens d'importer proprement des modules :

from module import fonction_1 fonction_2, ...

Avec cette syntaxe, on indique explicitement les fonctions que l'on souhaite utiliser. C'est très clair pour le lecteur et le développeur. Il n'y a pas de risques de confusion ou de conflits de noms car on voit les fonctions que l'on importe.

Pour utiliser les fonctions, il suffit de les invoquer simplement par leur nom : fonction1(arguments).

```
1 from random import randint
2 a = randint(1,100)
```

import module

Cette syntaxe concise à l'import est aussi sans ambiguïté. Par contre, elle va imposer que chaque appel de fonction soit précédé du module en préfixe.

Pour utiliser les fonctions, on les invoquera en faisant : module.fonction1(arguments).

```
1 import random
2 a = random.randint(1,100)
```

Il est possible en cas de nom de module trop compliqué de lui donner un *alias* plus cours par commodité :

```
1 import random as rd
2 a = rd.randint(1,100)
```

4. Exercices

Écrire un module aires permettant de calculer les aires de figures géométriques usuelles :

- triangle
- disque
- carré
- rectangle
- parallélogramme
- losange

Vous veillerez à respecter les consignes suivantes :

- à chaque figure correspondra une fonction du même nom
- vous déciderez des paramètres pertinents à communiquer à chaque fonction pour le calcul de l'aire
- le module et chaque fonction seront correctement documentées afin qu'un développeur ne connaissant pas votre module puisse l'utiliser facilement grâce à la fonction help.

Vous écrirez ensuite un petit programme afin de tester le fonctionnement de ce module.