

# **Structure d'un programme - modules**

# Table des matières

<b>I. Notion de module (package)</b>	<b>3</b>
<b>II. Exercice : Quiz</b>	<b>4</b>
<b>III. Imports de modules</b>	<b>4</b>
<b>IV. Exercice : Quiz</b>	<b>7</b>
<b>V. Création de son propre module</b>	<b>8</b>
<b>VI. Exercice : Quiz</b>	<b>9</b>
<b>VII. Essentiel</b>	<b>10</b>
<b>VIII. Auto-évaluation</b>	<b>10</b>
A. Exercice .....	10
B. Test .....	11
<b>Solutions des exercices</b>	<b>11</b>

## I. Notion de module (package)

### Contexte

En Python, les modules sont simplement des fichiers avec l'extension « *.py* » contenant du code Python qui peut être importé dans un autre programme Python. En termes simples, nous pouvons considérer qu'un module est identique à une bibliothèque de code ou à un fichier contenant un ensemble de fonctions que vous souhaitez inclure dans votre application. À l'aide de modules, nous pouvons organiser des fonctions, des classes ou tout bloc de code associés dans le même fichier. Ainsi, il est considéré comme une bonne pratique lors de l'écriture de codes plus volumineux pour des projets de niveau de production en Data Science de diviser les grands blocs de code Python en modules. Le module contient des définitions et implémentations des classes, des variables, et les fonctions pouvant être utilisées dans un autre programme.

Python offre de nombreux modules par défaut qui vous permettent d'aller au-delà des fonctionnalités. Un module est comme une extension : il vous donne accès à d'autres méthodes spécialisées. On regroupe dans un même module des fonctions sur la même thématique, ce sont des bibliothèques.

On appelle « *module* » tout fichier composé de code Python importé dans un autre fichier ou script.

Les modules permettent une séparation et donc une meilleure organisation du code. En effet, il est fréquent dans un projet de scinder son code en différents fichiers qui contiendront des parties cohérentes du programme final pour faciliter la maintenance, la compréhension générale du code et le travail en équipe.

Il existe trois grandes catégories de modules :

- Les modules standard qui ne font pas partie du langage lui-même mais sont automatiquement intégrés par Python.
- Les modules développés par des développeurs externes que nous pourrions utiliser.
- Les modules que nous développerons nous-mêmes.

Dans tous les cas, la procédure d'utilisation d'un module sera la même.

### Les modules standard à connaître

Ces modules vont étendre le langage et nous permettre d'effectuer toutes sortes d'opérations, notamment grâce aux fonctions qu'ils nous fournissent.

Pour importer un module Python, nous utiliserons à nouveau simplement une instruction d'importation comme si nous importions l'un de nos modules.

Les modules standards Python à connaître sont les suivants :

- Le module CGI (« *Common Gateway Interface* », ou « *Interface de Passerelle Commune* » en français) fournit des éléments permettant aux programmes Python de s'exécuter sur des serveurs http.
- Le module `datetime` fournit des classes pour une manipulation simple ou plus complexe des dates et heures.
- Le module `json` permet l'encodage et le décodage des données au format JSON.
- Le module `math` fournit un ensemble de fonctions permettant d'effectuer des calculs mathématiques complexes.
- Le module `OS` fournit un moyen portable d'utiliser les fonctionnalités dépendant du système d'exploitation.
- Le module `pickle` est utilisé pour sérialiser les objets Python.
- Le module `random` implémente des générateurs de nombres pseudo-aléatoires pour différentes distributions.
- Le module `RE` fournit des opérations d'expression régulière similaires à celles trouvées en Perl.
- Le module `socket` donne accès à l'interface sockets qui correspond à un ensemble standardisé de fonctions de communication.

- Le module sys donne accès à certaines variables système utilisées et maintenues par l'interpréteur, et à des fonctions qui interagissent fortement avec lui.
- Les modules urllib.request et urllib.parse permettent d'ouvrir, de lire et d'analyser des URL.

Vous pouvez trouver la liste complète des modules Python standard sur le site officiel.

## Exercice : Quiz

[solution n°1 p.13]

### Question 1

Qu'est-ce qu'un module Python ?

- ☐ Un fichier contenant des fonctions réutilisables
- ☐ Un environnement de travail additionnel pour Python
- ☐ Un environnement modulaire

### Question 2

Un module est forcément natif de Python.

- ☐ Vrai
- ☐ Faux

### Question 3

Un module permet une meilleure organisation du code.

- ☐ Vrai
- ☐ Faux

### Question 4

Quel module contient les fonctions mathématiques complexes liées à Python ?

- ☐ Le module math
- ☐ Le module sin
- ☐ Le module complex

### Question 5

En Python, il est nativement possible d'interagir avec le protocole http.

- ☐ Vrai
- ☐ Faux

## III. Imports de modules

## La méthode import

Comme nous le savons, le shell interactif Python possède un certain nombre de fonctions intégrées. En tant que démarrage du shell, ces fonctions sont chargées automatiquement et sont toujours disponibles, telles que :

- `print()` et `input()` pour les E/S
- Les fonctions de conversion de nombres telles que `int()`, `float()`, `complex()`
- Les conversions de types de données telles que `list()`, `tuple()`, `set()`, etc.

En plus de ces nombreuses fonctions intégrées, il existe également un grand nombre de fonctions prédéfinies disponibles dans le cadre des bibliothèques fournies avec les distributions Python. Ces fonctions sont définies dans des modules appelés « *modules intégrés* ».

Ces modules intégrés sont écrits en langage C et intégrés au shell Python.

Pour afficher une liste de tous les modules disponibles dans le langage de programmation Python, nous pouvons utiliser la commande suivante dans la console Python :

```
1 help('modules')
```

La sortie du code ci-dessus est illustrée ci-dessous :

```
>>>
>>> help("modules")

Please wait a moment while I gather a list of all available modules...

BaseHTTPServer      base64              io                  sgmlib
Bastion             bdb                itertools          sha
CGIHTTPServer       beaker             json               shelve
Canvas              binascii           keyword            shlex
ConfigParser        binhex             lib2to3            shutil
Cookie              bisect             linecache          signal
Dialog              bottle             linkedin           simplejson
DocXMLRPCServer     bs4                locale            site
FileDialog           bsddb              logging            six
FixTk               bz2                macpath            smtpd
HTMLParser          cPickle            macurl2path        smtplib
MimeWriter          cProfile           mailbox            sndhdr
Queue               cStringIO          mailcap            socket
ScrolledText        calendar           make               sqlite3
SimpleDialog        certifi            markupbase         sre
SimpleHTTPServer    cgi                markupsafe         sre_compile
SimpleXMLRPCServer  cgitb             marshal            sre_constants
SocketServer        chardet            math               sre_parse
StringIO            chunk              matplotlib          ssl
Tix                 cmath              md5                stat
Tkconstants         cmd                nhlib              statvfs
TkDnd               code               nimetools          string
Tkinter             codecs             nimetypes          stringold
UserDict            codeop             mimify             stringprep
UserList            collections        mmap               strop
UserString          colorsys           modulefinder       struct
_LWPCookieJar       commands           msilib             subprocess
_MozillaCookieJar  compileall         msvcrt             sunau
__builtin__         compiler           multifile          sunaudio
__future__          contextlib         multiprocessing    symbol
__abcoll            cookielib          mutex              symtable
__ast               copy               netrc              sys
__bisect            copy_reg           new                 sysconfig
__bsddb             ctypes             nose                tabnanny
__codecs            curses             nt                  tarfile
__codecs_cn         cyclar             ntpath              telnetlib
__codecs_hk         datetime           nturl2path          tempfile
__codecs_iso2022    dateutil           numbers             tempita
__codecs_jp         dbhash             oauthlib            test
__codecs_kr         decimal            opcode              test_files
__codecs_tw         decorator          operator            textwrap
__collections       difflib            optparse            this
__csv               dircache           os                  thread
__ctypes            distutils          os2emxpath          threading
__ctypes_test       django             panda               time
__elementtree       doctest            parser              timeit
__functools         dumbdbm            pdb                 tkColorChooser
__hashlib           dummy_thread       pickle              tkCommonDialog
__heapq             dummy_threading    pickletools         tkFileDialog
__hotshot           easy_install       pickletools         tkFont
__io                _io                _pickletools        tkMessageBox
__json              _json              _pickletools        tkSimpleDialog
```

Maintenant, discutons de certains des modules intégrés utiles et fréquemment utilisés de Python.

- Module de mathématiques
- Modules statistiques

Lorsque vous souhaitez utiliser les fonctionnalités du module math, celles-ci ne sont pas incluses et disponibles directement dans votre interpréteur. Il s'agit d'un module qu'il va falloir importer. Nous allons donc découvrir une première syntaxe d'importation.

Certains modules sont déjà installés sur votre ordinateur ! Il vous suffit de les activer en utilisant le mot-clé import suivi du nom du module.

Par exemple, le module math, contient les définitions de nombreuses fonctions mathématiques telles que sinus, cosinus, tangente, racine carrée, etc. Pour pouvoir utiliser ces fonctions, il vous suffit d'incorporer la ligne suivante au début de votre script.

```
1 import math
```

La syntaxe est facile à retenir : le mot-clé `import`, qui signifie importer en anglais, suivi du nom du module, ici `math`.  
Après l'exécution de cette instruction, rien ne se passe en apparence. En réalité, Python vient d'importer le module `math`. Toutes les fonctions mathématiques contenues dans ce module sont maintenant accessibles.

### Exemple

Pour assigner à la variable `racine`, la racine carrée de nombre.

```
1 racine = sqrt(nombre)
```

Pour assigner à la variable `sinusx`, le sinus de l'angle (en radians).

```
1 sinusx = sin(angle)
```

### Une autre méthode d'importation : `from ... import ...`

Il existe une autre méthode d'importation qui ne fonctionne pas tout à fait de la même façon. En fonction du résultat attendu, vous pouvez utiliser l'une ou l'autre de ces méthodes. Reprenons notre exemple du module `math`. Admettons que nous ayons uniquement besoin, dans notre programme, de la fonction renvoyant la valeur absolue d'une variable. Dans ce cas, nous n'allons importer que la fonction, au lieu d'importer tout le module.

```
1 from math import fabs
2 fabs (-5)
```

### Renommer les modules

Il est possible de modifier les noms des modules et leurs fonctions dans Python en utilisant le mot-clé « `as` ».

Vous voudrez peut-être changer un nom parce que vous avez déjà utilisé le même nom pour autre chose dans votre programme, un autre module que vous avez importé utilise également ce nom, ou vous voudrez peut-être abréger un nom plus long que vous utilisez beaucoup.

La construction de cette instruction ressemble à ceci :

```
1 import [module] as [another_name]
```

Modifions le nom du module mathématique. Nous allons changer le nom du module de `maths` en `m` afin de l'abréger. Notre programme modifié ressemblera à ceci :

```
1 import math as m
2 print(m.pi)
3 print(m.e)
```

Dans le programme, nous nous référons maintenant à la constante `pi` comme `m.pi` plutôt que `math.pi`.

Pour certains modules, il est courant d'utiliser des alias. La documentation officielle du module `matplotlib.pyplot` demande l'utilisation de `plt` comme alias :

```
1 import matplotlib.pyplot as plt
```

Cela permet aux programmeurs d'ajouter le mot plus court `plt` à n'importe laquelle des fonctions disponibles dans le module, comme dans `plt.show()`.

## Exercice : Quiz

[solution n°2 p.14]

### Question 1

Quel est le mot-clé permettant d'importer un module ?

- ☐ import
- ☐ save
- ☐ from

Question 2

Lors de l'import, si celui-ci est réussi vous êtes notifiés par Python.

- ☐ Vrai
- ☐ Faux

Question 3

Pour utiliser un module, il suffit de taper le nom de la fonction ou variable à l'intérieur pour l'utiliser.

- ☐ Vrai
- ☐ Faux

Question 4

On doit toujours importer toutes les fonctions d'un module.

- ☐ Vrai
- ☐ Faux

Question 5

Quelle est la syntaxe permettant d'importer seulement les fonctions choisies depuis un module ?

- ☐ import <nom de la fonction> from <nom du module>
- ☐ import <nom de la fonction> and <nom de la deuxième fonction>
- ☐ from <nom de la fonction> import <nom du module>

## V. Création de son propre module

Il serait judicieux de réutiliser une fonction bien écrite dans un autre programme Python. Vous avez la possibilité de le faire en créant vos propres modules.

Les modules sont regroupés autour d'un thème précis, on y rassemble les fonctions qui seraient réutilisables.

### Créer un module

Il est facile de créer ses propres modules en Python. Vous pouvez écrire vos fonctions et constantes dans un fichier que vous pourrez enregistrer comme n'importe quel script Python avec l'extension .py.

Par exemple, nous pouvons créer ici un module que nous allons enregistrer dans le fichier « *message.py* ».

```
1 """Module inutile qui affiche des messages"""
2 DATE = 16092008
3 def bonjour(nom):
4     """Dit Bonjour."""
5     return "Bonjour " + nom
6 def ciao(nom):
7     """Dit Ciao."""
```



```

8     return "Ciao " + nom
9 def hello(nom):
10     """Dit Hello."""
11     return "Hello " + nom

```

À noter que les trois guillemets en tête du module et de la fonction sont facultatifs au fonctionnement du module. Toutefois, ils jouent un rôle essentiel puisqu'ils font office de documentation ici.

### Utiliser ses propres modules

À noter que pour utiliser une fonction ou une variable issue d'un module, le fichier avec l'extension .py doit se trouver dans le répertoire courant (celui dans lequel on travaille) ou dans un répertoire listé par la variable d'environnement (il s'agit de PYTHONPATH dans votre système d'exploitation).

Ensuite vous pouvez importer de façon classique votre module avec la commande import pour utiliser ses fonctions et constantes. À noter que le module est bien enregistré avec l'extension .py, toutefois il n'est pas nécessaire de préciser cette extension pour l'importer. Finalement, vous pouvez utiliser les fonctionnalités de ce module de façon classique.

```

1 import message
2 message.hello("Joe")
3 'Hello Joe'
4 message.ciao("Bill")
5 'Ciao Bill'
6 message.bonjour("Monsieur")
7 'Bonjour Monsieur'
8 message.DATE
9 16092008

```

#### Méthode

Si vous utilisez un système d'exploitation basé sur UNIX (Mac et Linux), vous pouvez exécuter dans une console Bash cette commande qui vous permet de modifier la variable d'environnement PYTHONPATH :

```
1 export PYTHONPATH=$PYTHONPATH:/chemin/du/module
```

Si vous utilisez Windows, vous pouvez effectuer la même chose dans une console PowerShell avec cette commande :

```
1 env:PYTHONPATH += "C:\chemin\vers\du\module"
```

À noter que sous Windows, pour indiquer un chemin nous utilisons les antislashes (\).

Quand vous avez effectué cette manipulation, assurez-vous que le chemin du répertoire contenant vos modules a été ajouté à la variable d'environnement PYTHONPATH avec les commandes suivantes :

Pour Mac et Linux :

```
1 echo $PYTHONPATH
```

Pour Windows :

```
1 echo $env:PYTHONPATH
```

#### Fondamental

Si c'est la première fois que le module est importé, Python va créer un répertoire nommé `__pycache__` qui contiendra un fichier avec une extension .pyc. Ce fichier contient le bytecode, c'est le code précompilé du module.

## Exercice : Quiz

[solution n°3 p.14]

### Question 1

Que faut-il faire pour créer son module ?

- ☐ Écrire des fonctions ou variables dans un fichier.py
- ☐ Écrire des fonctions ou variables dans un fichier.py et le déclarer à Python
- ☐ Écrire des fonctions ou variables dans un fichier.py, le déclarer à Python et l'envoyer sur un dépôt

Question 2

On peut écrire un module Python n'importe où.

- ☐ Vrai
- ☐ Faux

Question 3

Il est obligatoire de mentionner l'extension .py lors de l'import d'un module.

- ☐ Vrai
- ☐ Faux

Question 4

Le mot-clé « *echo* » permet de déclarer une variable ?

- ☐ Vrai
- ☐ Faux

Question 5

Quelles sont les commandes nécessaires pour déclarer un module dans la variable PYTHONPATH en fonction de votre OS ?

- ☐ export PYTHONPATH=\$PYTHONPATH:/chemin/du/module
- ☐ env:PYTHONPATH += "C:\chemin\vers\du\module"
- ☐ PYTHONPATH = /chemin/du/module

## VII. Essentiel

Vous savez maintenant ce qu'est et comment utiliser un module Python. Vous avez également appris comment en créer un. Les modules Python sont indissociables de la programmation orientée objet. En effet, ils permettent une meilleure lisibilité et maintenabilité du code source. Il devient beaucoup plus facile de diviser son code en blocs et de l'organiser en suivant certaines normes. Cela permet de pouvoir contribuer à des projets open source en modifiant des modules utilisés par de grosses applications, par exemple.

## VIII. Auto-évaluation

### A. Exercice

Vous savez maintenant ce qu'est un module, comment l'importer mais aussi comment créer le vôtre. Il est donc temps de créer votre module.

#### Question 1

[solution n°4 p.16]

Vous avez créé un module sur votre bureau, pour une raison x vous voulez que le fichier reste sur votre bureau. Cependant votre dossier de projet se situe dans votre répertoire utilisateur. Comment allez-vous l'importer ?

**Question 2**

[solution n°5 p.16]

Lors du premier import de votre module vous avez pu constater l'apparition d'un dossier `__pycache__`, quelle est son utilité ?

**B. Test****Exercice 1 : Quiz**

[solution n°6 p.16]

## Question 1

Remplacez le trou : un module Python est un fichier avec l'extension \_\_\_\_\_ qui contient du code Python valide.

- ☐ .module
- ☐ .py
- ☐ pymodule
- ☐ .pym

## Question 2

Laquelle de ces définitions décrit correctement un module ?

- ☐ Conception et mise en œuvre de fonctionnalités spécifiques à intégrer dans un programme
- ☐ Définit la façon dont il doit être utilisé
- ☐ Tout programme qui réutilise du code

## Question 3

Lequel des éléments suivants n'est pas un avantage de l'utilisation de modules ?

- ☐ Fournit un moyen de réutiliser le code du programme
- ☐ Fournit un moyen de répartir les tâches
- ☐ Fournit un moyen de réduire la taille du programme
- ☐ Fournit un moyen de tester des parties individuelles du programme

## Question 4

Il est possible de traiter du JSON en Python.

- ☐ Vrai
- ☐ Faux

## Question 5

On peut donner n'importe quel nom à son module Python.


- ☐ Vrai
- ☐ Faux

**Solutions des exercices**




**Exercice p. 4 Solution n°1****Question 1**

Qu'est-ce qu'un module Python ?

- ☒ Un fichier contenant des fonctions réutilisables
- ☐ Un environnement de travail additionnel pour Python
- ☐ Un environnement modulaire
-  Un module est un fichier .py contenant des variables ou fonction réutilisables dans Python.


**Question 2**

Un module est forcément natif de Python.

- ☐ Vrai
- ☒ Faux
-  Un module peut être natif de Python ou bien avoir été développé par une tierce personne.


**Question 3**

Un module permet une meilleure organisation du code.

- ☒ Vrai
- ☐ Faux
-  Un module favorise la segmentation du code et donc une meilleure organisation de ce dernier.


**Question 4**

Quel module contient les fonctions mathématiques complexes liées à Python ?

- ☒ Le module math
- ☐ Le module sin
- ☐ Le module complex
-  C'est le module math qui contient les fonctions mathématiques complexes.

**Question 5**


En Python, il est nativement possible d'interagir avec le protocole http.

- ☐ Vrai
- ☒ Faux
-  Pour interagir avec ce protocole, il faudra installer le module CGI.

## Exercice p. 7 Solution n°2


### Question 1

Quel est le mot-clé permettant d'importer un module ?

- ☒ import
- ☐ save
- ☐ from
-  On importe un module Python grâce au mot-clé « *import* ».


### Question 2

Lors de l'import, si celui-ci est réussi vous êtes notifiés par Python.

- ☐ Vrai
- ☒ Faux
-  Lors de l'import, même si celui-ci est réussi rien ne se passe, du moins en apparence.


### Question 3

Pour utiliser un module, il suffit de taper le nom de la fonction ou variable à l'intérieur pour l'utiliser.

- ☒ Vrai
- ☐ Faux
-  Par exemple, une fois le module math importé, il suffit de taper sin pour utiliser la fonction sin.


### Question 4

On doit toujours importer toutes les fonctions d'un module.

- ☐ Vrai
- ☒ Faux
-  Il existe une syntaxe permettant d'importer seulement quelques fonctions d'un module.

### Question 5


Quelle est la syntaxe permettant d'importer seulement les fonctions choisies depuis un module ?

- ☐ import <nom de la fonction> from <nom du module>
- ☐ import <nom de la fonction> and <nom de la deuxième fonction>
- ☒ from <nom de la fonction> import <nom du module>
-  C'est la syntaxe « *from <nom de la fonction> import <nom du module>* » qui permet d'importer seulement certaines fonctions d'un module.

## Exercice p. 9 Solution n°3


**Question 1**

Que faut-il faire pour créer son module ?

- ☒ Écrire des fonctions ou variables dans un fichier.py
- ☐ Écrire des fonctions ou variables dans un fichier.py et le déclarer à Python
- ☐ Écrire des fonctions ou variables dans un fichier.py, le déclarer à Python et l'envoyer sur un dépôt
-  Il suffit d'écrire des fonctions et des variables dans un fichier .py.


**Question 2**

On peut écrire un module Python n'importe où.

- ☐ Vrai
- ☒ Faux
-  On peut, en soit, l'écrire n'importe cependant pour l'importer il faut qu'il soit dans notre répertoire courant ou déclaré dans la variable PYTHONPATH.


**Question 3**

Il est obligatoire de mentionner l'extension .py lors de l'import d'un module.

- ☐ Vrai
- ☒ Faux
-  Il n'est pas nécessaire de le mentionner lors de l'import du module.


**Question 4**

Le mot-clé « *echo* » permet de déclarer une variable ?

- ☐ Vrai
- ☒ Faux
-  La commande « *echo* » est utilisée dans différents langages informatiques et elle permet d'afficher une ligne ou alors la valeur d'une variable.

**Question 5**

Quelles sont les commandes nécessaires pour déclarer un module dans la variable PYTHONPATH en fonction de votre OS ?

- ☒ `export PYTHONPATH=$PYTHONPATH:/chemin/du/module`
- ☒ `env:PYTHONPATH += "C:\chemin\vers\du\module"`
- ☐ `PYTHONPATH = /chemin/du/module`
-  Il faut utiliser les deux premières commandes : l'une pour Linux, l'autre pour Windows.

**p. 10 Solution n°4**

Tout simplement en utilisant la variable PYTHONPATH qui contient tous les modules ou extensions dont Python doit avoir connaissance peu importe leur localisation sur la machine.

Pour un environnement Linux ou Mac :

```
1 env:PYTHONPATH += "C:\chemin\vers\du\module"
```

Pour un environnement Windows :

```
1 env:PYTHONPATH += "C:\chemin\vers\du\module"
```

**p. 11 Solution n°5**

Le dossier \_\_pycache\_\_ contient le code du module précompilé. Vous constaterez qu'à chaque fois que vous importez un nouveau module, Python se charge de le précompiler et de le stocker dans le dossier \_\_pycache\_\_, vous y trouverez le nom de votre module avec l'extension .pyc.

**Exercice p. 11 Solution n°6**

**Question 1**

Remplacez le trou : un module Python est un fichier avec l'extension \_\_\_\_ qui contient du code Python valide.

- ☐ .module
- ☒ .py
- ☐ pymodule
- ☐ .pym
- ☐ L'extension du fichier sera .py.

**Question 2**

Laquelle de ces définitions décrit correctement un module ?


- ☒ Conception et mise en œuvre de fonctionnalités spécifiques à intégrer dans un programme
- ☐ Définit la façon dont il doit être utilisé
- ☐ Tout programme qui réutilise du code
- ☐ Un module permet la conception et mise en œuvre de fonctionnalités spécifiques à intégrer dans un programme.

**Question 3**

Lequel des éléments suivants n'est pas un avantage de l'utilisation de modules ?

- ☐ Fournit un moyen de réutiliser le code du programme
- ☐ Fournit un moyen de répartir les tâches
- ☒ Fournit un moyen de réduire la taille du programme
- ☐ Fournit un moyen de tester des parties individuelles du programme



 Plus on importe de modules, plus on augmente la taille du bundle final.


#### Question 4

---

Il est possible de traiter du JSON en Python.

☒ Vrai

☐ Faux

 Il est possible de traiter du JSON, mais pour cela il faut installer le module json de Python.


#### Question 5

---

On peut donner n'importe quel nom à son module Python.

☒ Vrai

☐ Faux

 Il n'y a pas de convention de nommage sur les modules Python, vous pouvez donc lui donner le nom que vous souhaitez.