



Python

Gestion des modules

Par Richard BONNAMY



Introduction

Python- Définition

- Un **module** est un **fichier Python** (.py)
- Il est possible de **réutiliser** le code contenu dans un module grâce à un mécanisme d'**import**
- Les modules permettent une **meilleure organisation** du code.
- En Python, on distingue **trois catégories de modules**:
 - Les modules standards intégrés au langage Python
 - Les modules développés par des communautés
 - Les modules qu'on va développer nous-mêmes.
- Dans tous les cas, le mécanisme d'import est le même.

Python- Module

- Un programme Python est généralement composé d'un **module principal** qui importe différents modules (i.e. différents fichiers Python).
- Pour importer le code situé dans un module **nom_module.py**, on utilise la syntaxe suivante:

import nom_module

- Pour utiliser une ressource du module dans notre script, il faut préfixer le nom de la ressource par le nom du module et un point.
- Cela permet d'éviter les conflits dans le cas où on aurait des éléments de même nom dans différents modules.
- Exemple de **module bonjour**:

```
nom = "Pierre"

def dire_bonjour():
    print("Bonjour", nom)
```

Python- Module

- Attention cela ne fonctionne que si le fichier **nom_module.py** se situe dans le même répertoire que le module principal
- Dans le cas d'un module situé dans un sous-répertoire, on utilise la syntaxe suivante :

import sous-repertoire.nom_module

Python- Import

- Une fois le module importé, on accède aux **variables** et **fonctions** du module en préfixant le nom de la ressource par le nom du module.
- Exemple d'import du module bonjour précédent :

```
import bonjour  
  
print(bonjour.nom)  
bonjour.dire_bonjour()
```

Python- Import avec alias

- On peut préciser un **alias** du nom du module avec la syntaxe suivante :

`import nom_module as nom_alias`

- Une fois qu'un alias est défini, le nom du module n'est plus utilisable

- **Exemple :**

```
import bonjour as b  
  
print(b.nom)  
b.dire_bonjour()
```

Python- Import d'une ressource spécifique

➤ Il est possible **d'importer une ressource spécifique**.

➤ Pour cela, on utilise la syntaxe suivante :

from nom_module **import** nom_ressource

➤ La ressource est alors utilisable sans être préfixée du nom du module.

➤ Exemple :

```
from bonjour import dire_bonjour  
  
print(nom)  
dire_bonjour()
```

➤ Dans l'exemple ci-dessus, nom n'est plus accessible car non importée.

Python- Import de toutes les ressources

- Il est possible **d'importer toutes les ressources d'un module avec la syntaxe suivante :**

from nom_module **import** *

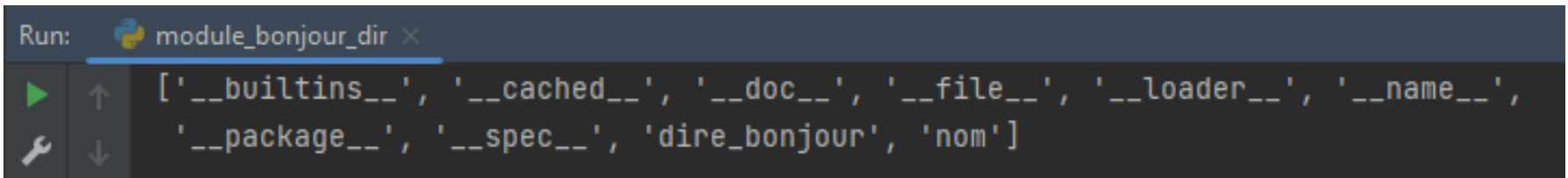
- L'ensemble des ressources sont alors utilisables sans être préfixées du nom du module.
- Exemple :

```
from bonjour import *  
  
print(nom)  
dire_bonjour()
```

Python- Lister les ressources d'un module

- Il est possible **de lister l'ensemble des ressources d'un module** avec la fonction **dir** qui renvoie **la liste des ressources**.
- Exemple :

```
import bonjour  
  
liste_ressources = dir(bonjour)  
print(liste_ressources)
```



Run: module_bonjour_dir ×


```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',  
 '__package__', '__spec__', 'dire_bonjour', 'nom']
```

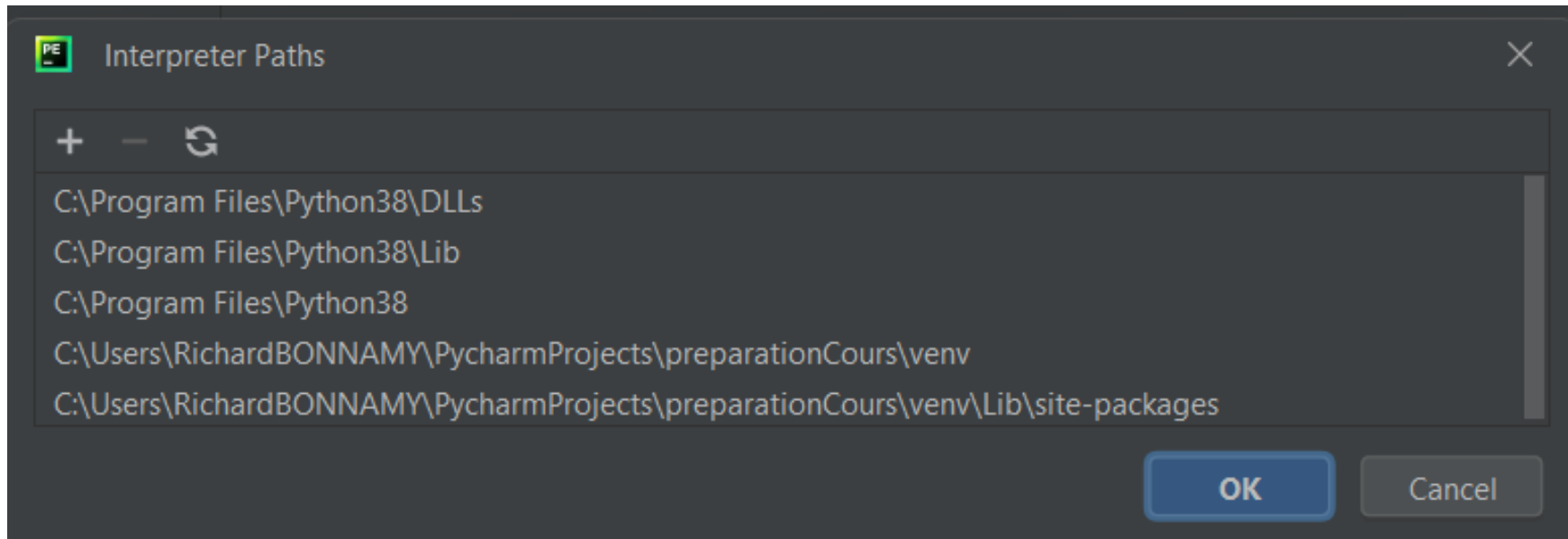
Python- Packages et installation

- Un **package** est constitué d'un **ensemble de modules**.
- La plupart des "bibliothèques" Python sont des packages : Pandas, TensorFlow, Django, etc.
- Les principaux packages sont référencés sur <https://pypi.org>
- Un outil comme **pip** permet d'installer un package en ligne de commande.
- Cet outil est intégré aux distributions Python depuis la version 3.4
- Syntaxe :

pip install nom_package

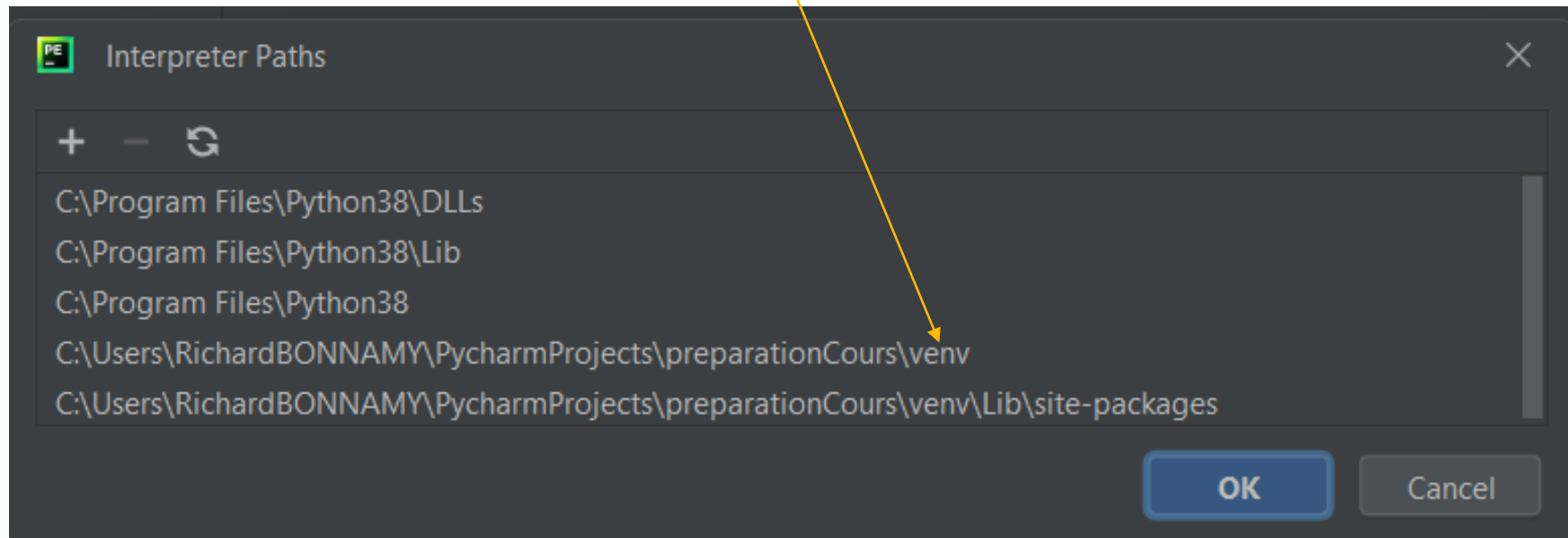
Python- Import et Path

- Lorsque l'interpréteur rencontre une instruction import, il parcourt le path pour rechercher le module.
- Le path, ou chemin de recherche, est une liste de répertoires dans lesquels l'interpréteur recherche les modules.
- Exemple sous PyCharm : (File > Settings > clic sur la sélection du Python interpréter > Show All > )



Python- Environnement virtuel

- Avec **PyCharm**, chaque projet est isolé dans un environnement virtuel
- Chaque environnement virtuel contient ses propres modules et packages
- Si vous installez un package, seul le projet courant aura ce package.
- Exemple d'environnement **venv** sous PyCharm :



Python- Pourquoi un environnement virtuel ?

- La **création d'un environnement virtuel est vivement** conseillée pour chaque projet, surtout si vous installez des packages.
- **Sans environnement virtuel**, chaque package est installé au niveau du répertoire d'installation du langage Python.
 - Si vous faites cela, chaque package installé deviendra une dépendance de chacun de vos projets Python.
 - En plus du nombre de dépendances qui peut devenir ingérable, vous pouvez avoir des problèmes de version de packages incompatibles.
- **PyCharm** crée automatiquement un environnement virtuel pour chaque nouveau projet.
- Si vous êtes en dehors de PyCharm, en ligne de commande :

python -m venv nom_repertoire_venv

En général on utilise venv comme nom de répertoire

Python- Que fait la création d'un environnement virtuel ?

- En supposant que vous ayez choisi **venv** comme nom de répertoire

python -m venv venv

- Cette commande a pour effet:

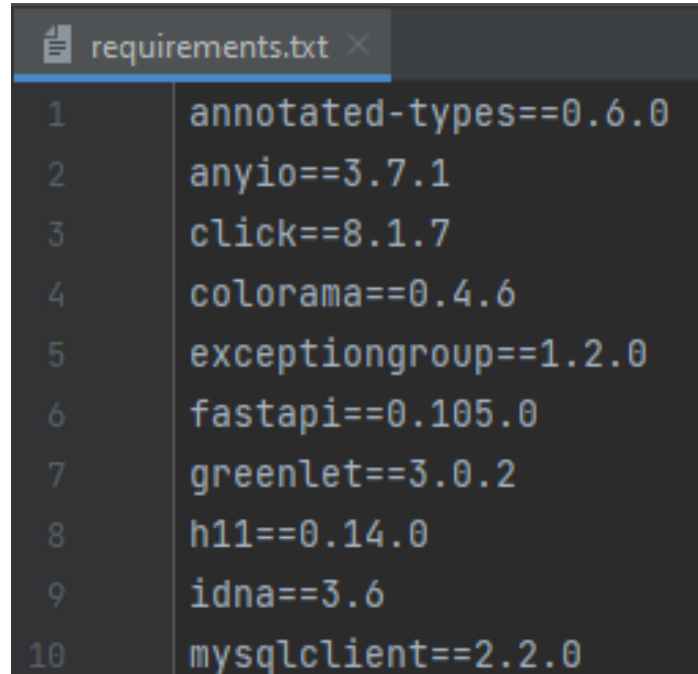
- De créer un répertoire nommé **venv** à la racine de votre projet
- De **créer une copie du langage Python** dans le répertoire:
 - **venv/Scripts** (sous Windows)
 - **venv/bin** (sur MacOS)
- Si vous êtes sous Windows et que vous n'utilisez pas **PyCharm** il faudra double-cliquer sur le fichier **venv/Scripts/Activate.bat** pour activer cette version de Python.
- Une fois l'activation effectuée, les commandes **pip install** que vous exécuterez installeront vos packages dans **venv/Lib**

Python- Comment conserver la liste des packages installés ?

- Avec un **environnement virtuel**, les packages sont installés dans **venv**
- Or, il n'y a pas de fichier de configuration qui liste les packages installés dans **venv**
- De plus **venv** est exclu des ressources à commiter.
- Comment conserver les dépendances de mon projet ? Une personne qui récupérerait mon projet sur GitHub en aurait besoin.

Python- Comment conserver la liste des packages installés ?

- Commande **pip freeze > requirements.txt**
- Cette commande génère à la racine de votre projet un fichier listant tous les packages nécessaires au fonctionnement de votre projet.

A screenshot of a code editor window titled 'requirements.txt'. The window displays a list of 10 packages and their versions, each on a new line. The packages are: annotated-types==0.6.0, anyio==3.7.1, click==8.1.7, colorama==0.4.6, exceptiongroup==1.2.0, fastapi==0.105.0, greenlet==3.0.2, h11==0.14.0, idna==3.6, and mysqlclient==2.2.0. The lines are numbered 1 through 10 on the left side of the editor.

```
1 annotated-types==0.6.0
2 anyio==3.7.1
3 click==8.1.7
4 colorama==0.4.6
5 exceptiongroup==1.2.0
6 fastapi==0.105.0
7 greenlet==3.0.2
8 h11==0.14.0
9 idna==3.6
10 mysqlclient==2.2.0
```

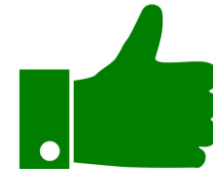
- **Ce fichier doit être comité !**

Python- Comment installer tous les packages à partir de requirements.txt ?

```
requirements.txt X
1 annotated-types==0.6.0
2 anyio==3.7.1
3 click==8.1.7
4 colorama==0.4.6
5 exceptiongroup==1.2.0
6 fastapi==0.105.0
7 greenlet==3.0.2
8 h11==0.14.0
9 idna==3.6
10 mysqlclient==2.2.0
```

- Si quelqu'un récupère votre projet, il n'a plus qu'à utiliser la commande suivante:

pip install -r requirements.txt





Installation de bibliothèques

Python- Commande pip

- La commande **pip** intégrée au langage Python permet d'installer des packages.

- Exemple :

pip install pygame

- Cette commande installe la dernière version de pygame

- Pour installer une version particulière d'un package il faut préciser la version avec ==

- Exemple :

pip install pygame==2.5.2



Modules Math, Random et Statistics

Python- Module math

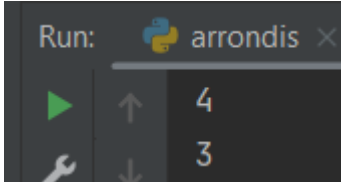
- Le module **math** fournit les fonctions mathématiques de base comme sinus, cosinus, tangente, logarithme ou exponentielle.
- Les **fonctions** les plus **couramment utilisées** sont :
 - Les fonctions **ceil()** et **floor()** renvoient l'arrondi du nombre passé en paramètre en arrondissant respectivement à l'entier supérieur et inférieur
 - La fonction **fabs()** renvoie la valeur absolue d'un nombre passé en paramètre
 - La fonction **isnan()** renvoie True si le nombre passé en paramètre est **NaN** (Not a Number ou pas un nombre en français) ou False
 - La fonction **exp()** permet de calculer des exponentielles
 - La fonction **log()** permet de calculer des logarithmes
 - La fonction **sqrt()** permet de calculer la racine carrée d'un nombre
 - Les fonctions **cos()**, **sin()** et **tan()** permettent de calculer des cosinus, sinus et tangentes et renvoient des valeurs en radians.

Python- Gestion des modules

- Le module **math** définit également des **constantes mathématiques** comme pi ou e, accessibles via `math.pi` et `math.e`.

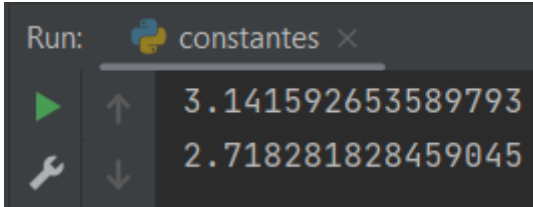
```
import math

print(math.ceil(3.1))
print(math.floor(3.1))
```



```
import math

print(math.pi)
print(math.e)
```



- Documentation complète : [la documentation](#).

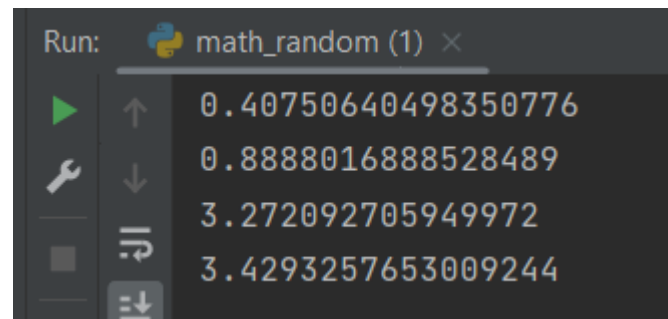
Python- Module random

- Le module **random** fournit des outils pour générer des nombres pseudo-aléatoires de différentes manières.
- La fonction **random()** est la plus utilisée.
 - Elle génère un nombre réel aléatoire dans la plage [0.0, 1.0[.
- La fonction **uniform()** génère un nombre réel aléatoire compris dans un intervalle. On lui passe deux nombres en paramètres :
 - le premier nombre représente la borne basse de l'intervalle
 - le second représente la borne supérieure.
 - Cette fonction se base sur random().

```
from random import random
from random import uniform

print(random())
print(random())

print(uniform(3, 4))
print(uniform(3, 4))
```



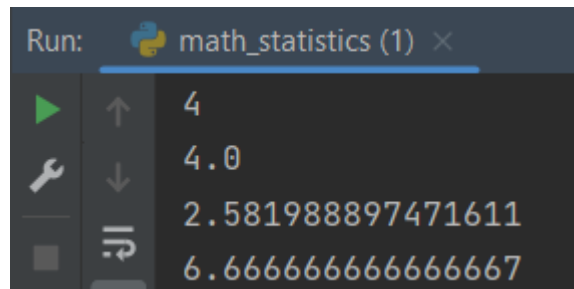
The screenshot shows a terminal window titled "Run: math_random (1) x". It displays the output of the Python code: two random numbers between 0.0 and 1.0, and two uniform random numbers between 3 and 4.

Function	Output
random()	0.40750640498350776
random()	0.8888016888528489
uniform(3, 4)	3.272092705949972
uniform(3, 4)	3.4293257653009244

Python- Module Statistics

- Le module **statistics** fournit des outils de calculs statistiques peu complexes, comme des calculs de moyenne, de médiane ou de variance.
- Ce module contient notamment les fonctions suivantes :
 - La fonction **mean()** permet de calculer une moyenne
 - La fonction **median()** permet de calculer une médiane
 - La fonction **stdev()** permet de calculer un écart type. Concrètement c'est la moyenne des écarts à la moyenne.
 - La fonction **variance()** permet de calculer une variance => mesure de la dispersion des valeurs d'un échantillon. Concrètement c'est la moyenne des carrés des écarts à la moyenne.

```
from statistics import *  
  
liste = [1, 3, 5, 7]  
print(mean(liste))  
print(median(liste))  
print(stdev(liste))  
print(variance(liste))
```



```
Run: math_statistics (1) ×  
4  
4.0  
2.581988897471611  
6.666666666666667
```



Modules Datetime, Time et Calendar

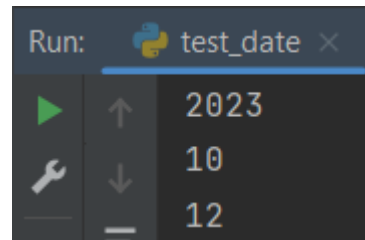
Python- Module datetime

- Le module **datetime** est le module de référence pour manipuler les dates et le temps.
- Le module **datetime** définit les types suivants :
 - **date** : représente une date (year, month et day)
 - **time** : représente un temps (hour, minute, second, microsecond et timezone)
 - **datetime** : date + temps (year, month, day, hour, minute, second, microsecond et timezone)
 - **timedelta** : représente la différence entre deux objets datetime en microsecondes
 - **timezone** : classe qui implémente la classe abstraite .

Python- Gestion des modules

- Création d'un objet **date** et affichage des attributs **year**, **month** et **day**

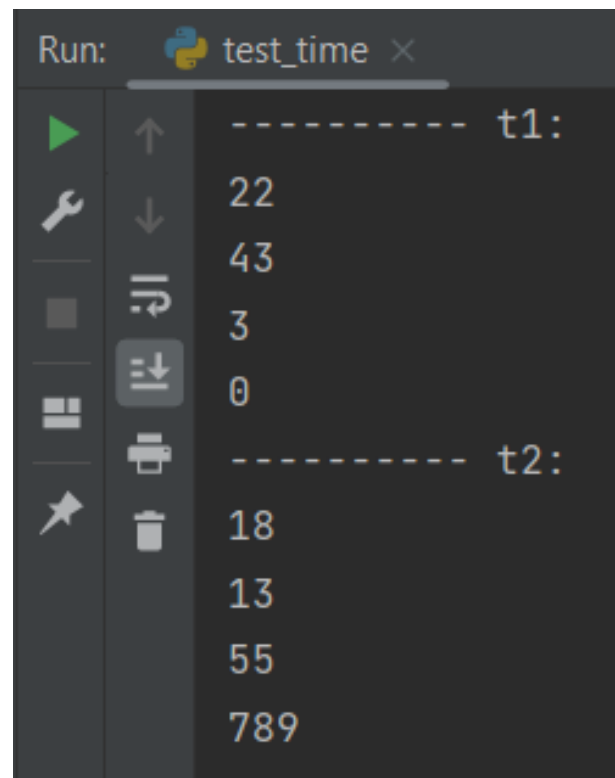
```
from datetime import *  
  
d = date(2023, 10, 12)  
print(d.year)  
print(d.month)  
print(d.day)
```



Python- Gestion des modules

- Création d'un objet **time** et affichage des attributs **hour**, **minute**, **second** et **microsecond**

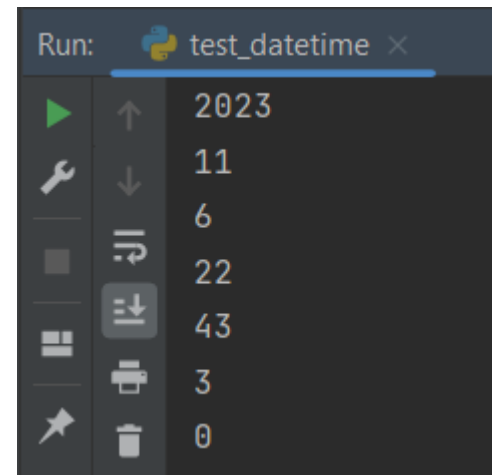
```
from datetime import *  
  
print("----- t1:")  
t1 = time(22, 43, 3)  
print(t1.hour)  
print(t1.minute)  
print(t1.second)  
print(t1.microsecond)  
  
print("----- t2:")  
t2 = time(18, 13, 55, 789)  
print(t2.hour)  
print(t2.minute)  
print(t2.second)  
print(t2.microsecond)
```



Python- Gestion des modules

- Création d'un objet **datetime** et affichage des attributs **year**, **month**, **day**, **hour**, **minute**, **second** et **microsecond**

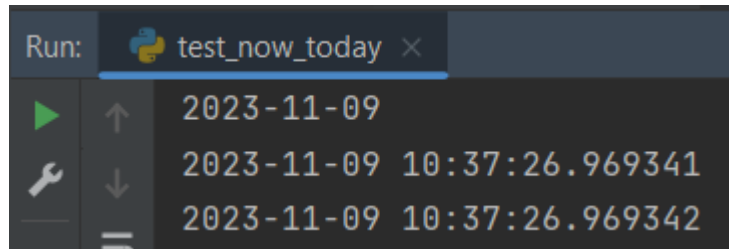
```
from datetime import *  
  
dt = datetime(2023, 11, 6, 22, 43, 3)  
print(dt.year)  
print(dt.month)  
print(dt.day)  
print(dt.hour)  
print(dt.minute)  
print(dt.second)  
print(dt.microsecond)
```



Python- Date et date/heure courante

- Utilisation des **méthodes de classe** **today()** et **now()**
- Seule la méthode **today()** existe pour la classe **date**
- Les méthodes **now()** et **today()** de la classe **datetime** semblent faire la même chose mais **now()** peut prendre en paramètre une timezone et non **today()**

```
from datetime import *  
  
print(date.today())  
print(datetime.now())  
print(datetime.today())
```



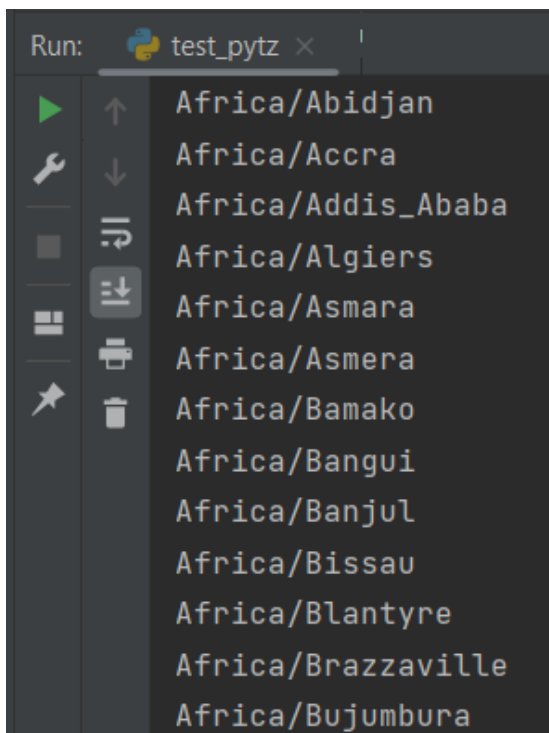
```
Run: test_now_today ×  
2023-11-09  
2023-11-09 10:37:26.969341  
2023-11-09 10:37:26.969342
```

Python- Le module pytz

- Le module **pytz** permet d'obtenir la liste des **fuseaux horaires (timezones)** utilisables
- **all_timezones** fournit la liste des 596 noms de **fuseaux horaires existants**

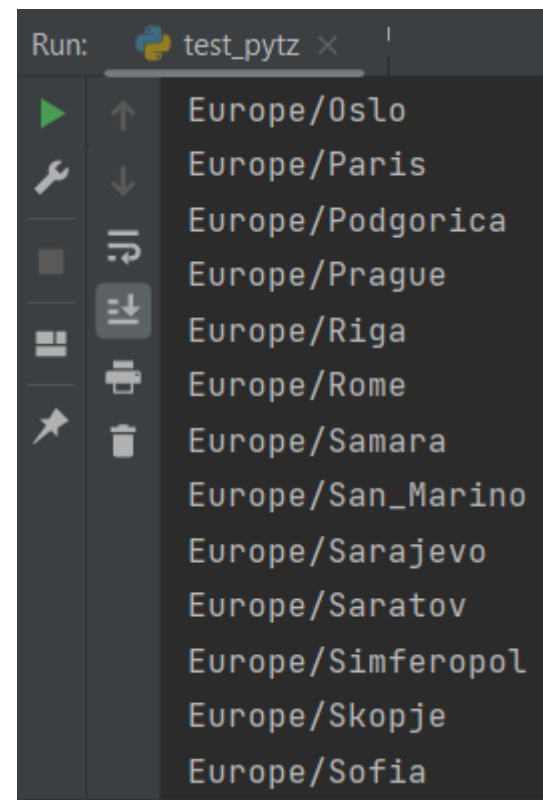
```
import pytz

for t in pytz.all_timezones:
    print(t)
```



Run: test_pytz x |

- ↑ Africa/Abidjan
- ↓ Africa/Accra
- ↺ Africa/Addis_Ababa
- ↻ Africa/Algiers
- ⇅ Africa/Asmara
- ⇅ Africa/Asmera
- ⇅ Africa/Bamako
- ✦ Africa/Bangui
- ✦ Africa/Banjul
- ✦ Africa/Bissau
- ✦ Africa/Blantyre
- ✦ Africa/Brazzaville
- ✦ Africa/Bujumbura



Run: test_pytz x |

- ↑ Europe/Oslo
- ↓ Europe/Paris
- ↺ Europe/Podgorica
- ↻ Europe/Prague
- ⇅ Europe/Riga
- ⇅ Europe/Rome
- ⇅ Europe/Samara
- ✦ Europe/San_Marino
- ✦ Europe/Sarajevo
- ✦ Europe/Saratov
- ✦ Europe/Simferopol
- ✦ Europe/Skopje
- ✦ Europe/Sofia

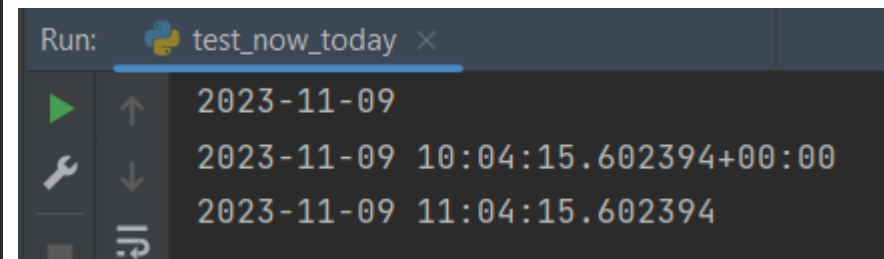
Python- Obtenir un objet timezone à partir de son nom

- Une fois le nom du fuseau horaire identifié, le module **pytz** fournit un moyen de récupérer une **instance de timezone** à partir de son **nom** :

```
from datetime import *
import pytz

timezoneParis = pytz.timezone("Europe/Dublin")

print(date.today())
print(datetime.now(timezoneParis))
print(datetime.now())
```



```
Run: test_now_today x
2023-11-09
2023-11-09 10:04:15.602394+00:00
2023-11-09 11:04:15.602394
```

- + 00:00 indique le décalage horaire par rapport au fuseau UTC.

Python- Parser une date ou datetime

- **Parser** signifie transformer une chaîne de caractères en objet.
- Ici il s'agit de transformer une chaîne de caractères contenant une date ou une date/heure en instance de datetime
- A noter que cette méthode n'existe que dans la classe datetime.
- Utilisation de la **méthode de classe strptime** avec 2 paramètres :
 - la chaîne de caractères,
 - le pattern de formatage de la chaîne

```
from datetime import *  
  
chaîne = '19/09/23 13:55:26'  
  
datetime = datetime.strptime(chaîne, '%d/%m/%y %H:%M:%S')  
  
print(type(datetime))  
print(datetime) # affichage au format ISO
```

Python- Patterns de formatage

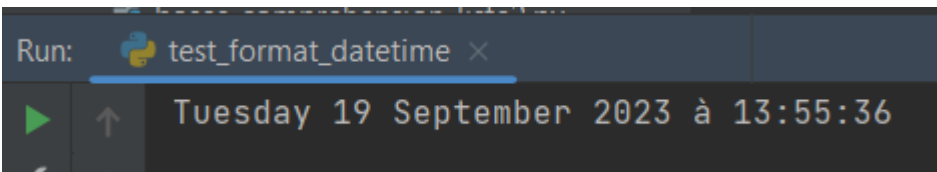
- Liste des patterns : <https://docs.python.org/3/library/datetime.html#strptime-and-strftime-behavior>
- Quelques patterns de base :

Pattern	Signification	Exemple
%d	Jour du mois sur 2 chiffres	De 01 à 31
%m	Mois sur 2 chiffres	De 01 à 12
%y	Année sur 2 chiffres, i.e. sans le siècle	De 00 à 99
%Y	Année sur 4 chiffres, i.e. avec le siècle	De 0001 à 9999
%H	Heure sur 24h	De 00 à 23
%I	Heure sur 12h	De 01 à 12
%M	Minutes	De 00 à 59
%S	Secondes	De 00 à 59
%f	Microsecondes	De 000000 à 999999
%j	Quantième (numéro de jour dans l'année)	De 000 à 366
%W	Numéro de semaine dans l'année	De 00 à 53
%a	Nom du jour sur 3 lettres (dépend de la locale)	Mon, Thu, ..., Sun en anglais
%A	Nom du jour complet (dépend de la locale)	Monday, Thursday, ..., Sunday en anglais
%b	Nom du mois sur 3 lettres (dépend de la locale)	Jan, Feb, Dec en anglais
%B	Nom du mois complet (dépend de la locale)	January, February, ..., December en anglais

Python- Formatage d'une date

- Il existe une méthode qui permet de **transformer** un objet **datetime** en **chaîne de caractères**.
- Il s'agit de la méthode `strftime(datetime)` de **datetime**
- Cet objet existe aussi pour les classes **date** et **time**.
- **Exemple :**

```
from datetime import *  
  
dt1 = datetime(2023, 9, 19, 13, 55, 36)  
  
print(dt1.strftime("%A %d %B %Y à %H:%M:%S"))
```



The screenshot shows a terminal window with the title "Run: test_format_datetime". The output of the code is "Tuesday 19 September 2023 à 13:55:36".

- Comment faire pour **changer les noms de jours et de mois** ?

Python- Utilisation du module locale

- Utilisation du **module locale** pour modifier les **paramètres culturels**.
- Les paramètres culturels permettent par exemple de traduire dans la langue de l'utilisateur les noms de jours et de mois.
- Exemple pour utiliser le français :

```
import locale
from datetime import *

locale.setlocale(locale.LC_ALL, 'French')
dt1 = datetime(2023, 9, 19, 13, 55, 36)

print(dt1.strftime("%A %d %B %Y à %H:%M:%S"))
```

- Il est également possible d'utiliser "fr_FR" ou "" pour laisser l'environnement décider de la langue.
- Les 3 exemples ci-dessous sont équivalents :

```
locale.setlocale(locale.LC_ALL, "fr_FR")
locale.setlocale(locale.LC_ALL, "French")
locale.setlocale(locale.LC_ALL, "")
```