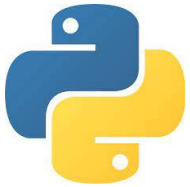


Introduction à la programmation en Python

M2 Risques et Environnement

UFR SEN – Université des Antilles

emmanuel.biabiany@univ-antilles.fr

Présentation du cours

2 séances de 2h de CM

- Introduction
- Données
- Algorithmique
  - Sélection
  - Répétition
- Fonctions
- Librairies
- Classes et dictionnaire
- Listes et tableaux
- Lecture et écriture de fichiers
- Graphiques

3 séances de 2h de TP

- Installation de python
- Présentation et installation Spyder
- TP d'application aux pluies
  - Rédaction d'un rapport

- Contrôle continue
  - Test de 30 min
  - Note du TP

Introduction

Langage libre et gratuit

→ Linux, Windows, Mac

https://www.python.org/downloads/

Interprétation → code

Terminal ou Fichier.py

Commentaire:

```
#sur une ligne
"""
 sur plusieurs ligne
"""
```

Opérateurs arithmétiques:

Addition (+), multiplication(*), soustraction(-), division(/), modulo(%)

Opérateurs logiques:

ET(and), OU(or), NON(not)

Valeurs logiques:

Vrai(True), Faux(False)

Données

en python → pas de déclaration

Affectation:

```
A = "Salut"
A = 3
```

Affichage:

```
print("message")
print(A)
print("message"+str(A))
```

Récupération saisies clavier:

```
A = input()
#possibilité d'avoir un prompt
A = input("message")
```

Attention à input()

→ renvoi toujours du `str`

Donc caster en type souhaité

ex., `int()`, `float()`

```
A = int(input("saisir age"))
```


Algorithmique

Sélection:

```
if condition :
        …
else:
        …
```

Condition:

Comparaison (<,>,<=,>=,!=)

Opération logique

Répétition simple:

```
for cpt in range(rep):
        …
```

Répétition simple:

```
while (condition):
        …
```


Fonctions

Action dédié → simplification programme

en python → pas de type

Déclaration:

```
def nom_fonction(param1,…, paramX=valX):
        …
        return var1, var2,…
```

Valeur de paramètre par défaut

Renvoie multiple

Appel:

```
A,B,… = nom_fonction(p1,…)
```

Déclaration avant appel

Fonction principale:

```
def main():
        …
if __name__ == "__main__":
        main()
```


Librairies (package)

en python → package

```
#importation
import nom_package
import nom_package as alias
import nom_package.sous_package
```

Package personnalisé

→ nom du fichier.py

Attention pas de main → fichier package

Packages science:

- Numpy,sympy
- Scipy, pandas
- Matplotlib

Packages système:

- Os, sys
- Shutil

Classes et dictionnaire

Modéliser objet → en python → class

Déclaration:

```
class nom_classe:
        prop1=val1
        def __init__(self,param1,…):
                prop1=param1
                …
        def __str__(self):
                return f"{self.pro1}"
        …
```

Usage:

```
#création variable
A = nom_classe(val1,…)
A.prop1 = val2
B = A.func(param1,…)

#affichage
print(A)
```

Classes et dictionnaire

Modéliser objet → en python → dict

Création objet dynamique

```
A = {"prop1":val1,…}
```

Attention

→Pas de constructeur (__init__)

→Pas de routine (__str__)

Usage:

```
A["prop1"] = val1
```

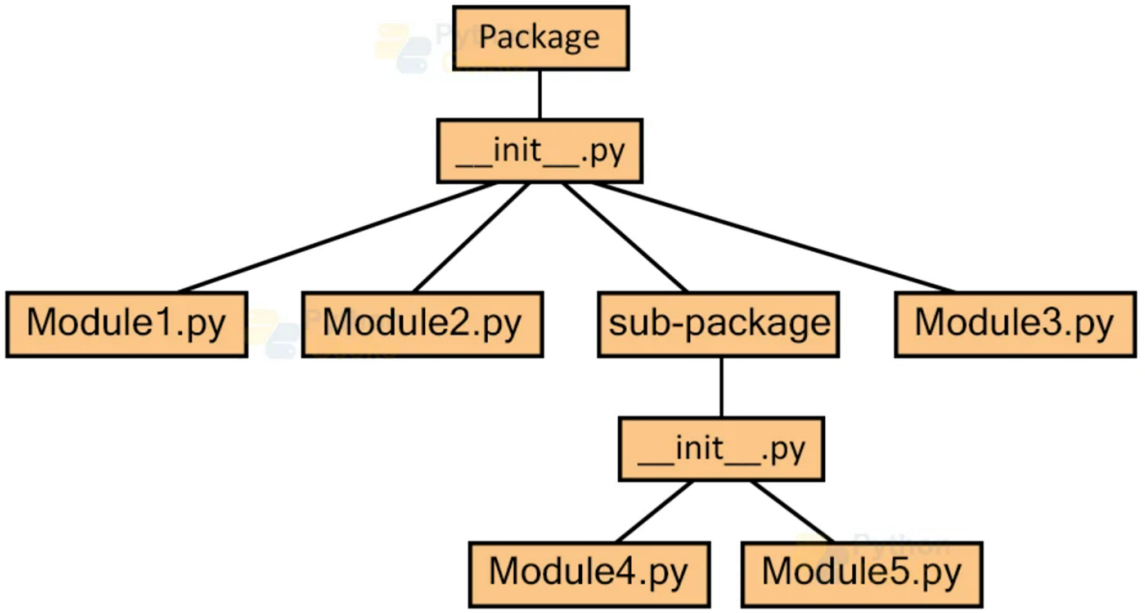
```
#affichage
print(A["prop1"])
```

Organisation d'un programme

```
#Librairies
import nom_package

#Classes
class nom_classe:

#Functions
def nom_fonction():
#Main
if __name__ == "__main__":
      main()
```



Listes et tableaux

en python → variable indicé → list

→ Ensemble de valeurs (peu importe la nature et le type)

→Taille variable

```
liste = []          liste.append(var)          var=liste.pop()
```

Usage:

```
liste = ["salut", 35, 'test', 8.9]
liste[0] = "Bonjour"; A = [1,2]; B = [3,4]; C = A + B → [1,2,3,4]
```

Autres fonctionnalités :

```
extend(), count(), reverse(), sort(), insert(), clear()
```


Listes et tableaux

en python → variable indicé → array

→Ensemble valeurs (de même nature)

→Taille fixe

Définition:

```
tab=numpy.empty(taille)
tab=numpy.empty((taille1,taille2))
tab=numpy.empty ((taille1,…,tailleN))
```

Voir aussi :

```
numpy.ones(), numpy.zeros()
numpy.sum(), .mean(), .max(), .min()
```

Usage:

```
temperatures = numpy.zeros(365)
notes = numpy.empty((25,3))
températures[0] = 25.5
notes[0,2] = 18.75
moyCC1 = numpy.mean(notes[:,0])
A = numpy.asarray([1,2])
B = numpy.asarray([3,4])
C = A + B
```

→ Opération entre tableau (+,-,*,/)

Lecture et écriture de fichiers

→ Chargement données: fichier (texte, .npy, autres) → tableau

```
data = numpy.loadtxt("nom_fichier.csv", dtype=float(), délimiter=";")
data = numpy.load(…)       #format bianaire optimisé
import pandas, netCDF4, xarray
data = pandas.read_excel("nom_fichier.xlsx")
data["nom_colonne3"] = data["nom_colonne1"] + data["nom_colonne2"]
data = netCDF4.Dataset("nom_fichier.nc") #format international
data["nom_var"][start1:end1,start2:end2,…]
data = xarray.open_dataset("nom_fichier.grib2")
data = data.to_dataframe()
```


Lecture et écriture de fichiers

→ Sauvegarde données: tableau → fichier (texte, .npy, autres)

```
numpy.savetxt("nom_fichier.csv", data, fmt="%f", sep=";")
numpy.save(…)
import pandas, xarray
pandas.to_excel("nom_fichier.xlsx", data)
pandas.to_csv("nom_fichier.csv", data)
pandas.to_hdf("nom_fichier.nc4", data)
xarray.to_grib("nom_fichier.grib2")
```