SPUS201 - UE SCIENCES : Introduction à la programmation 2

Denis Dubruel - Cours Magistral N°1

Année 2024/2025

UNIVERSITÉ CÔTE D'AZUR

courriel : prenom.nom@univ-cotedazur.fr

Table des Matières.

Introduction

- **Définition (Larousse) :**
  - Programmation : Ensemble des activités liées à la définition, l'écriture, la mise au point et l'exécution de programmes informatiques; séquence des ordres auxquels doit obéir un dispositif.
- Pour coder nous choisissons un langage : Python 3
- Crée en 1989 par Guido Van Rossum

- Langage maintenu par la Python Software Foundation : `https://www.python.org` Version 3.13 (oct 2024)

Introduction

Ce cours nécessite en pré requis d'avoir suivi le cours du premier semestre :
SPUS100 - UE SCIENCES : Introduction à la programmation.

Table des matières

Environnement Python - IDE (Environnement Développement Intégré)

- IDE (Integrated Development Environment) : ensemble d'outils pour les programmeurs comportant a minima :
  - un éditeur de texte pour écrire le code.
  - des fonctions par clic pour éxécuter le code ligne à ligne.
  - un débogueur (debugger), pour aider à la correction d'erreur.
  - autres fonctions bien pratiques.

[Grace Hopper vers 1960]

Bug(ang) ou Boque (Fr) terme technique pour décrire les défauts de systèmes mécaniques datant d'au moins avant les années 1870. Thomas Edison l'utilisait dans ses notes.

[Panne signalée en 1947 par G. Hopper, des opérateurs ont trouvé après et collé la première punaise (bug en anglais) en écrivant "first actual case of bug being found".

Environnement Python - IDE (Environnement Développement Intégré)

- Thonny, IDE simple, disponible via : `https://thonny.org/` A installer sur vos ordinateurs. IDE Retenue pour cette UE

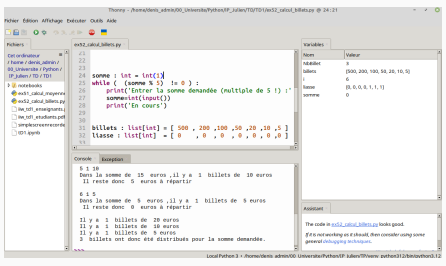


Table des matières

Manipuler les variables - rappel

Définition (rappel) : En informatique, les variables sont des espaces mémoire "élémentaires" qui associent un nom (l'identifiant pour le programmeur ou adresse pour l'ordinateur) à une valeur.

Bonne pratique (vue dans l'UE IP1) de définition explicite à respecter :

NomVar : type = Valeur initiale

Exemple de déclaration explicite :

programme.py

```
i : int=7                #variable i de type entier de valeur 7.
x : float= 3.2           #variable x de type float de valeur 3,2 .
ch : str='Toto'          # chaine de type string contenant 'Toto'.
LI : list[int]=[1,3,4] # liste d'entiers
L :  list[str]=['a','b','c']  # liste de chaines de caractère(s)
b : bool = True          # Booléen
```


Manipuler les variables -règles de nommage (PEP 8)

- Nomenclature complète https://peps.python.org/pep-0008/#naming-conventions
- Règles principales : (⚠pas d'espace dans le nom ! ⚠)
  - **Variables** (dont le contenu change !) : en lettres minuscules avec des underscores pour séparer les mots (ex : ma_variable ou ma_fonction) "snakecase".
  - **Constantes** : en lettres majuscules avec des underscores pour séparer les mots (ex : VITESSE_LIMITE).
  - **Classes** : commencent par **une majuscule, sans underscores**, ex : **Pixel , ImagePPM**, **PositionObjetMobile** ("Camel Case").
  - Attention aux mots réservés :

programme.py

```
print = 4   # definition de la variable print (osons !)
print(print) #  Surprise...
```


Manipuler les variables -Le zen de Python (PEP 20)

- Le PEP 20 (19/08/2004) reprend les principes généraux du codage (à lire en nuançant).
- traduction française sur : https://fr.wikipedia.org/wiki/Zen_de_Python

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one– and preferably only one –obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than *right* now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea – let's do more of those !

Manipuler les variables - Opérations

- Les 4 opérations sur entiers et flottants (+ , - , * , /)
- Elever à la puissance
- les opérations sur les entiers (division euclidienne)

SHELL

```
>>>2 ** 3 #   se lit 2 à la puissance 3 , ici 2x2x2=8
8
>>>1.44 ** 0.5   # racine carrée et oui !
1.2

15 // 4 # Quotient de la division entière d'un nombre
>>> 3
17 % 5 # Reste après la division avec
>>> 2
```

- 17 % 5 se lit 17 modulo 5 (...et vaut 2 car $17 = 3 \times 5 + 2$)

Comparer les variables

- pour tester une égalité, utiliser ==
- pour tester une différence, utiliser !=
- pour comparer <, >, <=, et >=

SHELL

```
>>>a : int = 1
>>>b : int = 2
>>> a == b # a est égal à b ?
False
>>> a != 4 #
a est il différent de 4 ?
True
>>> a > b #    a supérieur à b ?
False
>>> b < 2 # b inférieur à 2  ?
False
```

- La comparaison est vraie (True) ou fausse (False) : un objet de type booléen !
- Vu en IP1 pour les boucles :

programme.py

```
while i<len(chaine) :
        ...instuctions
        i=i+1
...
```


Table des matières

Manipuler les variables - Booléens

- Un booléen prend 2 valeurs : True ou False.

SHELL

```
>>> b : bool = False  # exemple de typage et affectation
>>> not(b)
        True
>>> 5 == 5      # condition toujours vraie .
    True
>>> not (5 == 5)     # exemple de négation d'une condition.
    False
```


Manipuler les variables - Booléens

- 3 opérations possibles :
  "négation (**not**)" "**OU** logique" "**ET** logique"
- Table de vérité :

Négation :

a	**not**(a)
True	False
False	True

Opération booléenne : OU / OR

- Table de vérité.

OU logique

a	b	a **OR** b
False	False	False
False	True	True
True	False	True
True	True	True

- ♡"a **OU** b faux" équivalent à "tous les deux sont faux".♡
- Utile pour les conditions plus complexes :

programme.py
```
while (0 < x  or  y < 0 ) :
        ...instructions
```


Opération booléenne : ET / AND

- Table de vérité.

ET logique :

a	b	a **AND** b
False	False	False
False	True	False
True	False	False
True	True	True

- ♡"a **ET** b" vrai équivalent à "tous les deux sont vrais".♡
- Utile pour les conditions plus complexes :

programme.py
```
while ( n<len(C)  and  x > 0 ) :
        ...instructions
```


Manipuler les variables -Exercices de Logique -Booléens

Compléter les tables :

a	b	a **AND** b	**not**(a **AND** b)
False	False		
False	True		
True	False		
True	True		

a	b	**not**(a)	**not**(b)	**not**(a) or **not**(b)
False	False			
False	True			
True	False			
True	True			

Conclusion ?

Manipuler les variables -Booléens -propriétés

SHELL
```
>>> not (a and b) == (not a) or (not b)     # Lois de Morgan
True
>>> not (a or b) == (not a) and (not b)     # Lois de Morgan :
True
```

SHELL
```
>>> (a and b) == (b and a)                  # Commutativité
True

>>> (a or b) or c == a or (b or c)          # Associativité
True

>>> a or (b and c) == (a or b) and (a or c)  # Distributivité
 True

>>>  a and (b or c) == (a and b) or (b and c)  # Distributivité
 True
```


Manipuler les variables -Exercices logique booléenne

Exercices d'entraînement à maîtriser :

1) Parmi les quatre expressions suivantes, laquelle s'évalue en **True** ?

- **Réponse A :** False and (True and False)
- **Réponse B :** False or (True and False)
- **Réponse C :** True and (True and False)
- **Réponse D :** True or (True and False)

2) Sachant que l'expression « **not(a or b)** » a la valeur **True**, quelles peuvent être les valeurs des variables booléennes a et b ?

- **Réponse A :** True et True
- **Réponse B :** False et True
- **Réponse C :** True et False
- **Réponse D :** False et False

3) Pour quelles valeurs booléennes des variables a, b et c l'expression « **(a or b) and (not c)** » a-t-elle pour valeur True

- **Réponse A :** a = True b = False c = True
- **Réponse B :** a = True b = False c = False
- **Réponse C :** a = False b = False c = False
- **Réponse D :** a = False b = True c = True

4) Choisir une expression booléenne pour la variable S qui satisfait la table de vérité suivante.

A	B	S
False	False	True
False	True	False
True	False	True
True	True	True

- **Réponse A :** A ou (non B)
- **Réponse B :** (non A) ou B
- **Réponse C :** (non A) ou (non B)
- **Réponse D :** non (A ou B)

5) On considère une formule booléenne **form** des variables booléennes a et b dont voici la table de vérité. Quelle est cette formule booléenne ?

A	B	form
True	True	False
False	True	False
True	False	True
False	False	False

- **Réponse A :** a and b
- **Réponse B :** a or b
- **Réponse C :** a and not(b)
- **Réponse D :** not(a) or b

Table des matières

Tests Conditionnels - utilisation des booléens

Ci dessous **condition_1** et **condition_2** sont des booléens!

programme.py
```
bloc # exécuté dans tous les cas car pas d'indentation.

if condition_1:      # un booléen !
        bla # exécuté si la condition 1 est vraie
elif condition_2:        # un booléen !
        ble # exécuté si la condition 1 est fausse
        ble # et la condition 2 est vraie
else:
        bli # exécuté si les deux conditions sont fausses


bly # exécuté dans tous les cas
bly # car il n'y a plus d 'indentations
```


Table des matières

Manipuler les variables -chaine de caractères

- ♡**Définition** (simple) : Une chaine de caractères est une séquence de symboles alphanumériques.

programme.py
```
ch : str ="Année 1968"    # definition de la chaine
print(ch)
```

SHELL
```
>>>
Année 1968
```


Manipuler les variables -chaine de caractères

- Concaténation avec les opérateurs + et * :

programme.py
```
ch : str ="Année 1968"    # definition de la chaine
ch2 = ch + "Toto"     # Concaténation
print(ch2)
ch3 = 2 * ch       # encore une concaténation
print()      # saut de ligne à l'affichage
print(ch3)
```

SHELL
```
>>>
Année 1968Toto

Année 1968Année 1968
```


Manipuler les variables -chaine de caractères

- Le caractère spécial « \ »
  (antislash (fr) ou backslash(ang)) permet :
  - d'insérer des codes spéciaux (sauts à la ligne, apostrophes, guillemets etc...

SHELL
```
>>>txt1 : str = '"C\'est beau !" dit-elle.'
>>>print(txt1)
"C'est beau !" dit-elle.

>>>txt2 : str = "'C\'est beau !' dit-elle."
>>>print(txt2)
'C'est beau !' dit-elle.
```


Manipuler les variables -chaine de caractères & méthodes

- ♡**Déf** :Les méthodes permettent d'analyser ou modifier des objets Python. Pour utiliser une méthode sur un objet on utilise la syntaxe : objet.méthode(*paramètre(s)*)
- str.lower()

programme.py
```
from string import *
txt4 : str ="Il y a du soleil à Brest dit Erwann."
print( txt4.lower())
```

SHELL
```
>>>
        'il y a du soleil à brest dit erwann.'
```


- str.upper()

programme.py
```
from string import *
txt4 : str ="Il y a du soleil à Brest dit Erwann."
print( txt4upper( ) )
```

SHELL
```
>>>
        'IL Y A DU SOLEIL À BREST DIT ERWANN.'
```


Manipuler les variables -chaine de caractères & méthodes

- str.capitalize()

programme.py
```
from string import *
txt4 : str ="Il y a du soleil à Brest dit Erwann."
print( capitalize( ) )
```

SHELL
```
>>>
'Il y a du soleil à brest dit erwann.' # Il ne reste que la première majuscule.
```


Manipuler les variables -chaine de caractères & méthodes

- Différence entre **str.isdigit())** et **str.isnumeric()** ???
- Bien savoir chercher dans la documentation Python pour les détails.
  - savoir définir/chercher un script de test avec un cas élémentaire.
  - faire varier les éléments pour bien comprendre les détails.

A vous de chercher pour vous convaincre par une démarche de recherche.

Manipuler les variables - autres méthodes

- La documentation officielle contient toutes les explications sur les méthodes pour manipuler les chaînes de caractères (45 en nov 2024 !!).
  https://docs.python.org/fr/3/library/stdtypes.html#string-methods

1. **str.capitalize()**
2. str.casefold()
3. str.center()
4. str.count()
5. str.encode()
6. str.endswith()
7. str.expandtabs()
8. str.find()
9. str.format()
10. str.format_map()
11. str.index()
12. str.isalnum()
13. str.isalpha()
14. str.isascii()
15. str.isdecimal()
16. str.isdigit()
17. str.isidentifier()
18. str.islower()
19. **str.isnumeric()**
20. str.isprintable()
21. str.isspace()
22. str.istitle()
23. str.isupper()
24. str.join()
25. str.ljust()
26. **str.lower()**
27. str.lstrip()
28. str.maketrans()
29. str.partition()
30. str.replace()
31. str.rfind()
32. str.rindex()
33. str.rjust()
34. str.rpartition()
35. str.rsplit()
36. str.rstrip()
37. **str.split()**
38. str.splitlines()
39. str.startswith()
40. str.strip()
41. str.swapcase()
42. str.title()
43. str.translate()
44. **str.upper()**
45. str.zfill()

Manipuler les variables -chaine de caractères

- Les chaînes sont immuables ou non modifiables. (il faut créer une nouvelle chaîne !)
- Les chaînes sont indexables (accès possible aux éléments).

programme.py

```
phrase : str='De gauche à droite !'
i : int = 0
endroit :str=""    # initialisation du résultat
envers : str=""
while i < len(phrase) : #
    envers= phrase[i] + envers
    endroit = endroit + phrase[i]
    i=i+1
print(envers)
print(endroit)
```


programme.py

```
phrase : str='De gauche à droite !'
i : int = 0
endroit :str=""    # initialisation du résultat
envers : str=""
while i < len(phrase) : #
    envers= phrase[i] + envers
    endroit = endroit + phrase[i]
    i=i+1
print(envers)
print(endroit)
```

SHELL

```
! etiord à ehcuag eD
De gauche à droite !
```


Table des matières

Entrées et Sorties : input()

- fonction input() :
  - Interrompt le programme.
  - Attend une saisie clavier terminée par <Entrée> ou <Enter>.
  - Renvoie une chaine de caractères.

programme.py

```
reponse : str =""
nb : int = 0
reponse = input("Entrez un nombre svp : ")
if reponse.isdigit() :
        nb = int(reponse)   # conversion de type
        print('Le double du nombre est : ', 2*nb)
else :
        print('erreur de saisie !')
```


programme.py

```
reponse : str =""
nb : int = 0
reponse = input("Entrez un nombre svp : ")
if reponse.isdigit() :
        nb = int(reponse)   # conversion de type
        print('Le double du nombre est : ', 2*nb)
else :
        print('erreur de saisie !')
```

SHELL

```
>>>
 Entrez un nombre svp : 25
 Le double du nombre est :  50
>>>
```


Entrées et Sorties : print()

- fonction **print()** :
  - Affiche la valeur de l'argument entre parenthèses.
  - Passe à la ligne.

SHELL

```
>>> print("Jean") # affiche du texte
    Jean

>>> x : int = 24
>>> print(x) # affiche le contenu d'une variable.
   24

>>> prenom : str = "Jean"
>>> nom :str = "Bon"
>>> print(prenom, nom, "a", x, "ans") # affichage de texte
                                      #et contenu de variables
   Jean Bon a 24 ans
```


Entrées et Sorties : print()

SHELL

```
>>> print(prenom,nom) # affiche et sépare d'un espace
                      # par défaut
Jean Bon

>>> print(prenom,nom, sep="_") # affiche les séparations
                    # avec le symbole  underscore
Jean_Bon
```

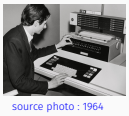

programme.py

```
print(prenom, end= ""  ) #le retour charriot est annulé
print(nom)      # tout sera affiché sur la même ligne.
```

SHELL

```
>>>
JeanBon
```

print (imprimer), vient de l'époque ou les périphériques de sortie étaient des imprimantes ex :



source photo : 1964 ordinateur IBM

Entrées et Sorties : f-strings

- f-strings : améliorent l'affichage des variables.
  - code plus lisible.
  - affichage formatté, plus lisible.

code.py

```
prenom : str = "Jean"
nom : str = "Bon"
print(prenom, "a", x, "ans") # affichage de texte
                             # et contenu de variables

print(f"{prenom} a {x} ans") #  le code est ici + lisible !
```

SHELL

```
Jean a 24 ans
Jean a 24 ans
```


Table des matières

Manipulation Fichiers *.txt - Lecture

- Avant l'informatique les données étaient stockées dans des fichiers papiers par exemple, album photos etc.
- Un fichier était le support physique (salle , meuble, classeur etc).Les fichiers contenaient des fiches (papier en général, microfilm etc).
- En informatique, un fichier est un ensemble de données numérique binaires sur un support physique (carte perforée, bande magnétique, disquette, disque dur, mémoire)
- On peut ouvrir, lire , écrire, fermer un fichier. On dit que l'on lit les données, ou charge (load) les données pour les traiter ensuite.

Manipulation Fichiers *.txt - Lecture

- la méthode **readlines()** renvoie une liste contenant **TOUTES** les lignes du fichier. Le caractère "\n" est le retour à la ligne.

code.py

```
from io import  TextIOWrapper
lignes : list [str] =[]
fichier : TextIOWrapper = open("fich_etu.txt","r", encoding="utf-8")# ouverture
lignes = fichier.readlines( ) #chargement de toutes les lignes
fichier.close( )    #fermeture.
print(lignes)       # affiche le contenu du fichier
```

SHELL

```
>>>
['Arthur   18\n', 'Adrien   20\n', 'Amélie   19\n', 'Léa     20\n']
```


Manipulation Fichiers *.txt - Ecriture

- Similaire à la lecture (⚠Retour charriot!!)

code.py

```
Etudiants : list[str] = ["Nicolas", "Eudes", "Clotaire"] # données
fichier: TextIOWrapper =open("Prenoms.txt", "w",encoding="utf-8")# ouverture
i:int =0
while i< len(Etudiants):
    fichier.write(Etudiants[i])  # boucle d'écriture
    i = i+1
fichier.close( ) # fermeture
```

Après avoir écrit le fichier, on vérifie le contenu. Les retours à la ligne sont manquants.

SHELL

```
>>> affiche_contenu("Prenoms.txt")
NicolasEudesClotaire
```


Manipulation Fichiers *.txt - Ecriture

code.py

```
Etudiants = ["Nicolas", "Eudes", "Clotaire"] # data
fichier : TextIOWrapper =open("LesPrenoms2.txt", "w",encoding="utf-8") # ouverture
i : int=0
while i< len(Etudiants):
    fichier.write(f'{Etudiants[i]}\n') # écriture avec ajout du \n
    i=i+1
fichier.close( )    # fermeture
```

SHELL

```
>>> affiche_contenu("LesPrenoms2.txt")
Nicolas
Eudes
Clotaire
```


Table des matières

Licence