Programmation Orientée Objet en Python

Syntaxe Python

T. Dietenbeck
thomas.dietenbeck@sorbonne-universite.fr

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Pour vous rassurer

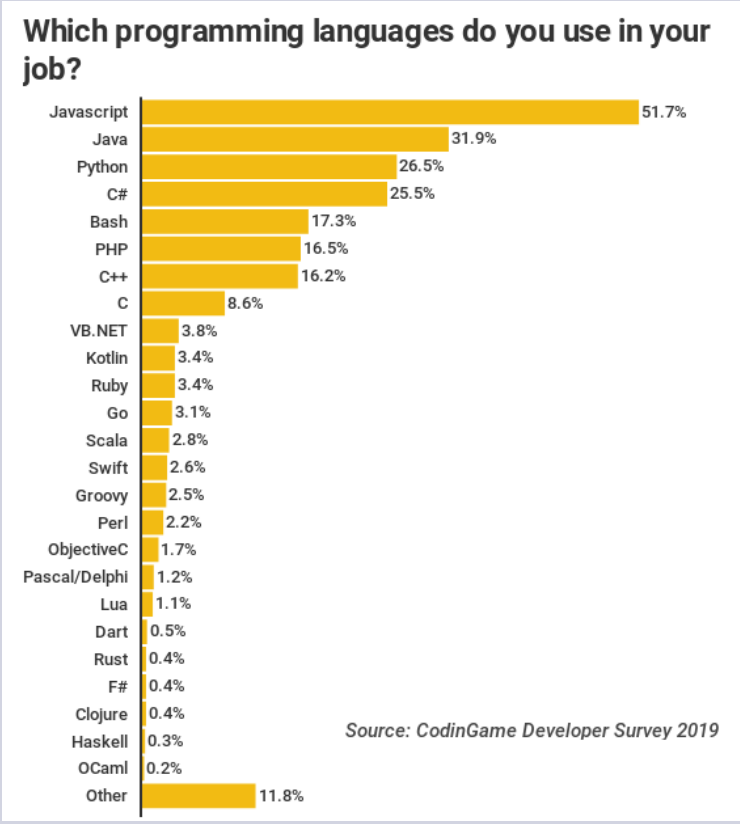
[Alan Perlis]

- "Programmer est un acte contre nature."
- "Dans un ordinateur, le langage naturel n'est pas naturel."
- "Il y aura toujours des choses que nous aimerions dire dans nos programmes, mais qui ne peuvent être que mal dites avec tous les langages connus."

- "Enseigner la programmation va à l'encontre de l'éducation moderne : Quel est le plaisir à planifier, se discipliner à organiser ses pensées, faire attention aux détails et apprendre à être autocritique ?"

Pourquoi Python ?

Langage utilisé (et apprécié) dans l'industrie



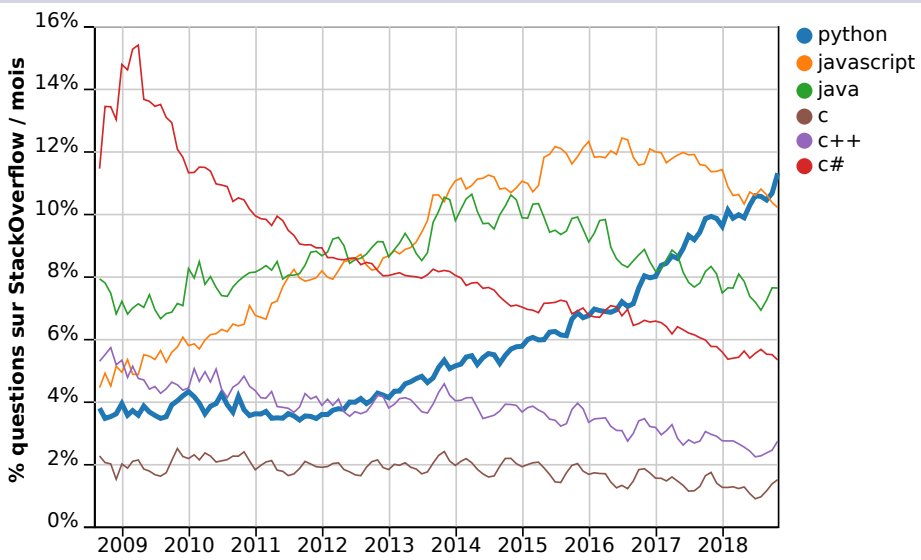
Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. Java		97.5
4. C		96.7
5. C#		89.4
6. PHP		84.9
7. R		82.9
8. JavaScript		82.6
9. Go		76.4
10. Assembly		74.1

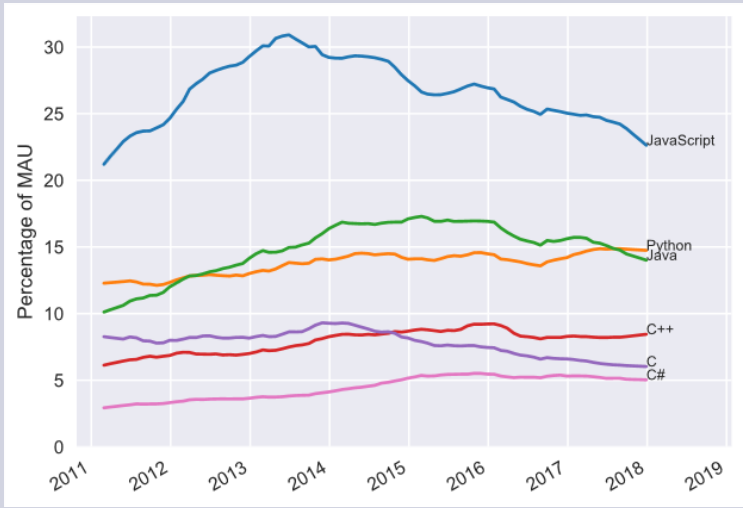
Classement IEEE des langages de programmation

SCIENCES SORBONNE UNIVERSITÉ

Pourquoi Python ?

Forte croissance et communauté active





Utilisateur mensuel actif sur Git

SCIENCES SORBONNE UNIVERSITÉ

De quoi ai-je besoin ?

Éditeur de texte

- Bloc-note, Geany, notepad++, ...
- Tout **SAUF** un traitement de texte (Word, OpenOffice)
- Favoriser les éditeurs avec surbrillance syntaxique (Geany, notepad++)
- **Le must :** éditeur intégré (Spyder, pyCharm, Jupyter notebook, ...)

Interpréteur

- Interpréteur Python 3
- **Documentation Python**
- **Le must (?) :** installer **Anaconda**
  - Python 3 + Interpréteur
  - Package manager permettant d'inclure facilement de nombreux modules scientifiques (numpy, matplotlib, tensorflow, pytorch, ...)

SCIENCES SORBONNE UNIVERSITÉ

Premier programme

Définition

- Le texte du code source Python est sauvegardé avec l'**extension .py**

Un premier programme : monPremierProg.py

```
print( "Vive les loutres!" )
print( "Et aussi Charlie" )
```

Premier programme

Définition

- Le texte du code source Python est sauvegardé avec l'**extension .py**

Un premier programme : monPremierProg.py

```
print( "Vive les loutres!" )
print( "Et aussi Charlie" )
```

Interprétation et exécution

Depuis une console (on suppose que le programme est présent dans le dossier courant)

`python3 monPremierProg.py`

Premier programme : console

Écriture à la console

L'affichage d'un résultat en Python se fait à l'aide des instructions :

- `print( message, end="")` : affiche le message
- `print( message )` : affiche le message et va à la ligne

Remarque : si le message contient un mélange de caractères et de valeurs, elles sont séparées par une ",". Python ajoute automatiquement un espace entre 2 valeurs

Exemple

```
print( "Vive les loutres", end = "" )  # Affichage simple
print( " et Charlie!" )     # Affichage + retour a la ligne

i = 1337
print( "Du texte, 1 variable:", i, " et 1 valeur:", 42 )
```

affichera :

```
Vive les loutres et Charlie!
Du texte, 1 variable: 1337 et 1 valeur: 42
```

SCIENCES SORBONNE UNIVERSITÉ

Premier programme : console

Lecture à la console

La lecture d'une valeur en Python se fait à l'aide de l'instruction :

- `input( message )` : affiche le message (facultatif) puis attend une valeur

Remarque : `input` renvoie une chaine de caractères. Il faut donc convertir les valeurs numériques vers le type souhaité (`int`, `float`)

Exemple

```
i = int( input( "Entrer un entier: " ) )    # Valeur entiere
f = float( input( "Entrer un reel: " ) )    # Valeur reelle
phrase = input( )                           # Chaine de caracteres
```

SCIENCES SORBONNE UNIVERSITÉ

Modules et packages

Définition

- Module :
  - ensemble de fonctions ou de classes regroupées dans un même fichier
- Package :
  - ensemble de modules regroupés dans un même répertoire
  - ces modules ont généralement un point commun (*e.g.* gestion des véhicules, voitures, camions, . . .)
  - le nom du dossier défini le nom du package

Objectif

- Améliorer la réutilisabilité d'un code
  - gain de temps : pas besoin de recoder certaines fonctions
  - gain en lisibilité : code plus court et regroupé par thème

Modules et packages

Syntaxe

Utilisation d'une classe ou d'un ensemble de classes existantes :

- `import random` : importe le package `random`
- `from math import cos` : importe la fonction acos du package `math`
- `from math import *` : importe toutes les fonctions du package `math`

Exemple

```
from random import random    # Generation de nombres aleatoires
import math                  # Fonctions mathematiques usuelles

nAlea = random( )    # Nombre aleatoire dans [0; 1[
angle = math.acos( nAlea )  # Angle correspondant
print("acos(", nAlea, ") =", angle, "rad" )
```

affichera

```
acos( 0.3016517887149436 ) = 1.264371655140188
```

Les erreurs

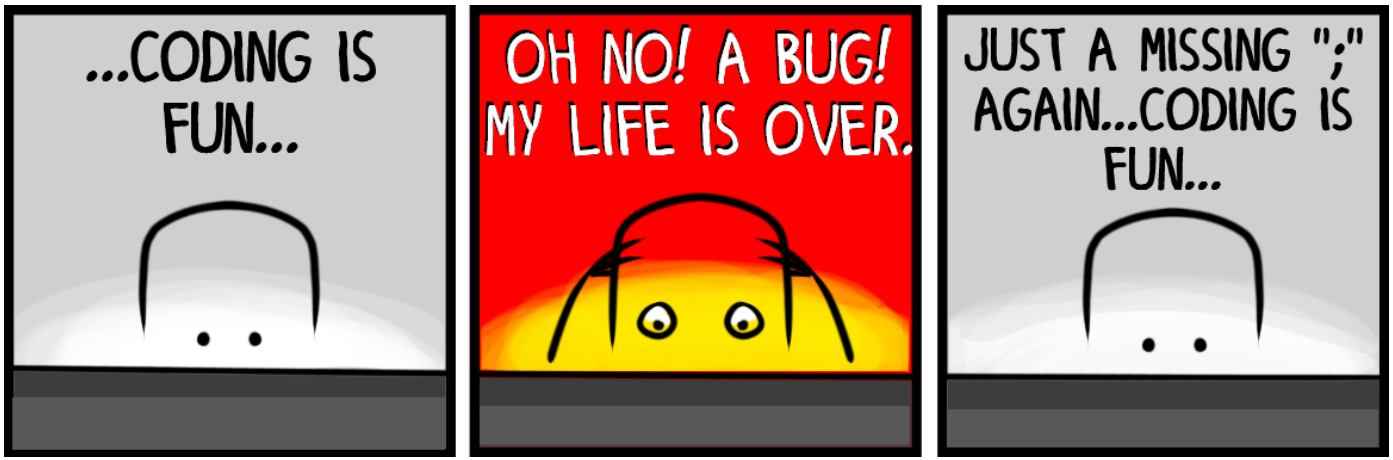
Définition

Il existe 2 types d'erreur

- les erreurs syntaxiques : le progamme ne compile pas
- les erreurs à l'exécution : le programme ne s'exécute pas correctement

[Alan Perlis]

"Il y a deux manières d'écrire des programmes sans erreurs ; seule la troisième marche."



SCIENCES SORBONNE UNIVERSITÉ

Erreurs syntaxiques

Définition

- Erreurs qui surviennent pendant la compilation du programme
- Non respect des règles de syntaxe (*e.g.* mauvaise indentation, variable non déclarée, . . .)
- Faciles à corriger : l'interpréteur les détecte et les signale

Exemple

```
x = 1
z = x + a
  print( "z =", z )
```

```
  File "main.py", line 2, in <module>
    z = x + a
NameError: name 'a' is not defined
```

```
 File "main.py", line 3
   print( "z =", z )
   ^
IndentationError: unexpected indent
```

Erreurs d'exécution

Définition

- Erreurs qui surviennent pendant l'exécution du programme :
  - soit le programme se bloque et affiche un message d'erreur
  - soit le programme produit un résultat faux
- Plus difficiles à corriger : le compilateur ne les détecte pas. Il faut exécuter le programme pas à pas pour comprendre où survient l'erreur

Erreurs classiques

- Avec un message d'erreur
  - Variable non créée (`AttributeError`)
  - Accès à une case hors du tableau (`IndexError`)
- Produisant un mauvais résultat
  - Variable mal initialisée
  - Mauvaise indentation
  - Condition toujours vraie / fausse $\Rightarrow$ pas de passage dans la boucle

Commentaires

Intérêt

- Permet d'expliquer
  - le rôle d'une variable
  - le fonctionnement d'une méthode
  - l'objectif d'une série d'instructions
- Ne sont pas compilés $\Rightarrow$ n'allourdit pas l'exécutable (n'ayez donc pas peur d'en écrire)
- **Remarque :** en Python, on utilise parfois des `"""chaines de caracteres"""` pour des commentaires longs (sur plusieurs lignes)

[Alan Perlis]

"La documentation est comme une assurance-vie : le bénéficiaire n'est presque jamais celui qui l'a signée."

Commentaires

Les 2 types de commentaires

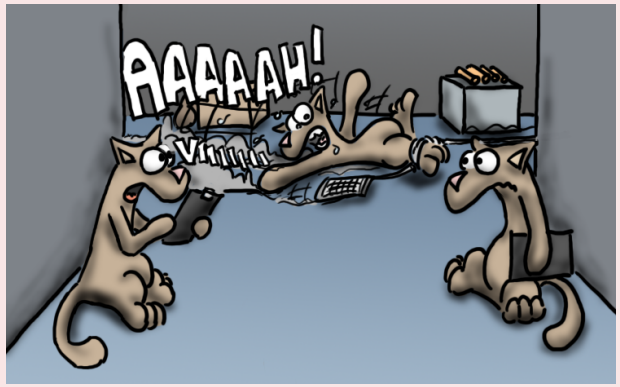
```
# Commentaire sur une ligne

def maMethode( param1 ) :
  """ Ceci est un exemple de documentation pour maMethode
      @param param1 le premier parametre de ma fonction
      @return True en cas de succes ou False en cas d'echec
      @author Thomas Dietenbeck
      @version 1.0
  """
```

Guide de Syntaxe

Ce qu'il ne faut surtout pas faire

```
def loutre( c, b, \
 a ) :
  if c<a : print(b)
  else: print(a)
  return "Facile"
pingouin, canard=41\
    ,1337; vache= 142857 if(pingouin  \
  > 40) else canard-pingouin
print( pingouin, """
      """, loutre(vache, canard, pingouin) )
```



[Martin Golding]

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."

Un guide appelé **Zen of Python** décrit quelques règles de bonne conduite

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Rappels

Qu'est ce qu'un programme ?

- des variables : pour stocker les résultats, des valeurs intermédiaires, *etc.*
- des instructions : les opérations (somme, lecture, *etc.*) à faire pour parvenir au résultat souhaité.

Définitions

- Variable : zone mémoire définie par
  - Nom : identificateur évocateur
  - Type : entier, caractère, booléen, ...
  - Valeur : son contenu
- Type
  - comparable aux unités de mesure en physique
  - domaine de valeurs, propriétés, opérateurs associés

$\Rightarrow$ On ne mélange pas les types

SCIENCES SORBONNE UNIVERSITÉ

Les types

Définition

- Python propose 4 types par défaut
  - les booléens `bool` (`True` ou `False`)
  - les entiers `int`
  - les réels `float`
  - les caractères et chaînes de caractères `str`

Remarques

- Python est un langage non typé. Le type d'une variable est choisi automatiquement et peut changer en cours d'exécution
- On peut obtenir le type d'une variable avec la fonction `type( maVariable )`

SCIENCES SORBONNE UNIVERSITÉ

Conversion de type (cast)

Définitions

- Conversion possible entre entiers et réels
- Deux types de conversion :
  - **Elargissante** : conversion d'un type vers un type plus large (implicite)
  - **Restrictive** : conversion d'un type vers un type plus petit (doit être explicite)

Exemple de conversions élargissantes

```
i = 42    # i est un entier
f = i + 0.1  # la valeur de i est convertie (automatiquement) en
    float puis ajoutee a 0.1
```

Exemple de conversions restrictives

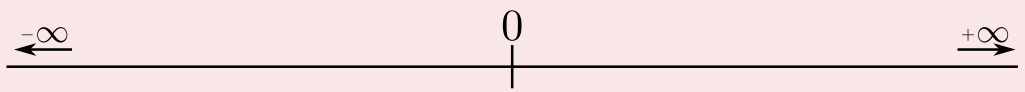
```
f = 13.37
# Attention a la perte de precision lors d'une conversion!
i = int( f )          # i = 13 (Valeur tronquee)
```

SCIENCES SORBONNE UNIVERSITÉ

Quelques dangers

La précision des réels

```
f = 0.1   # Pas de representation binaire exacte
          # => f est tronquee
          # et 3*f n'a pas la meme representation que 0.3
print( 3*f == 0.3 )   # => affiche False
```



En mathématiques

En informatique

SCIENCES SORBONNE UNIVERSITÉ

La liste des opérateurs

Opérateurs	Associativité
( ) [ ] .	Gauche à Droite
`not`	Droite à Gauche
**	Gauche à Droite
* / // %	Gauche à Droite
+ -	Gauche à Droite
< <= > >= `isinstance`	Gauche à Droite
== !=	Gauche à Droite
`and`	Gauche à Droite
`or`	Gauche à Droite
= += -= *= /= //= %=	Droite à Gauche

SCIENCES SORBONNE UNIVERSITÉ

Opérateurs d'affectation et arithmétiques

Opérateur d'affectation

Affectation simple	`a = 0`

Opérateurs arithmétiques

Addition	`a + b`
Soustraction	`a - b`
Multiplication	`a * b`
Division réelle	`a / b`
Quotient	`a // b`
Modulo	`a % b`
Puissance	`a ** b`

Exemples

```
unEntier = 8 + 3                # unEntier = 11
unEntier = unEntier % 7         # unEntier = 4

unReel = 3.                     # unReel = 3.0
unReel = unReel * unEntier      # unReel = 12.0
unReel = unReel / 5             # unReel = 2.4
```

SCIENCES SORBONNE UNIVERSITÉ

Opérateurs d'affectation et arithmétiques

Affectations élargies

Addition	`a += b` ⇒ `a = a + b`
Soustraction	`a -= b` ⇒ `a = a - b`
Multiplication	`a *= b` ⇒ `a = a * b`
Division réelle	`a /= b` ⇒ `a = a / b`
Quotient	`a //= b` ⇒ `a = a // b`
Modulo	`a %= b` ⇒ `a = a % b`
Puissance	`a **= b` ⇒ `a = a ** b`

- Ces notations permettent d'alléger / de condenser le code.

Exemples

```
unEntier = 8 + 3    # unEntier = 11
unEntier %= 7       # unEntier = 4

unReel = 3.         # unReel = 3.0
unReel *= unEntier  # unReel = 12.0
unReel /= 5         # unReel = 2.4
```

SCIENCES SORBONNE UNIVERSITÉ

Opérateurs relationnels et logiques

Opérateurs relationnels

Inférieur	`a < b` et `a <= b`
Supérieur	`a > b` et `a >= b`
Egalité	`a == b`
Inégalité	`a != b`

Opérateurs logiques

NON logique	`not a` (si a est un booléen !)
ET logique	`a and b`
OU logique	`a or b`

Exemples

```
val1, val2 = 1, 2
b = (val1 == 1) and (val2 > 2)
print( "(val1 = 1 ET val2 > 2) est ", b )
b = not (val1 == 1) or (val2 > 2)
print( "(val1 != 1 OU val2 > 2) est ", b )
```

SCIENCES SORBONNE UNIVERSITÉ

Plan

Structures de contrôle

Principe

- En règle générale, les instructions sont exécutées séquentiellement (dans le sens de lecture)
- Les structures de contrôle permettent de modifier cette ordre de lecture
  - soit en choisissant une suite d'instructions selon le résultat d'un test

    si $a > 2$ **alors**
    | $a \leftarrow a + 2$
    sinon
    | $a \leftarrow a * 2$
  - soit en répétant une suite d'instructions

    tant que $a > 2$ **faire**
    ⌊ $a \leftarrow a - 2$

[Alan Perlis]

"Un programme sans boucle et sans structure de donnée ne vaut pas la peine d'être écrit."

SCIENCES SORBONNE UNIVERSITÉ

Sélections : `if/else`

L'instruction `if`

Exécution d'une instruction ou d'un bloc d'instructions si une condition est remplie.

```
if condition :
  # instruction(s) a executer si la condition est vraie
  # Toutes les instructions ayant la meme indentation
  # seront executees
```

Remarques :

- si le test est faux, on ne fait rien.
- en Python l'indentation **définit** les instructions qui seront executées

Sélections : if/else

L'instruction if

Exécution d'une instruction ou d'un bloc d'instructions si une condition est remplie.

```
if condition :
  # instruction(s) a executer si la condition est vraie
  # Toutes les instructions ayant la meme indentation
  # seront executees
```

Remarques :

- si le test est faux, on ne fait rien.
- en Python l'indentation **définit** les instructions qui seront executées

La clause else

Définition d'instructions à exécuter si la condition n'est pas remplie.

```
if condition :
  # instruction(s) a executer si la condition est vraie
else :
  # instruction(s) a executer si la condition est fausse
```

SCIENCES SORBONNE UNIVERSITÉ

Sélections : if/else

if ... else en cascade

Plusieurs `if ... else` peuvent se succéder

```
if condition1 :
  # instruction(s) a executer si la condition 1 est vraie
elif condition 2 :
  # instruction(s) a executer si la condition 1 est fausse
  # et la condition 2 vraie
else :
  # instruction(s) a executer si les conditions 1 et 2
  # sont fausses
```

SCIENCES SORBONNE UNIVERSITÉ

Sélections : if/else

Exemple

```
if unEntier % 2 == 0 :
  print( unEntier, " est pair." )
else :
  print( unEntier, " est impair." )

if unReel > 0 :
  print( unReel, " est positif." )
elif unReel < 0 :  # Si on arrive ici, on sait que unReel est
    negatif ou nul
  print( unReel, " est negatif." )
else :
  print( unReel, " est nul." )
```

Sélections : if/else

Sélection en ligne

Plutôt que d'écrire

```
if condition :
  uneVar = valeurVraie
else :
  uneVar = valeurFausse
```

on peut utiliser la syntaxe

```
uneVar = valeurVraie if condition else valeurFausse
```

Exemple

```
dX = int( input( "Entrer un deplacement lateral: " ) )
gaucheOuDroite = "gauche" if dX < 0 else "droite"
print( "Vous avez entre", dX, end="" )
print( " et nous allons donc a", gaucheOuDroite )
```

SCIENCES SORBONNE UNIVERSITÉ

Les boucles

Définitions

- On écrit une seule fois une séquence d'instructions qui pourra être exécutée plusieurs fois
- 2 façons de répéter la séquence
  - **Nombre de répétitions non connu a priori** : boucle avec condition d'arrêt
    `while` : la condition est déterminable avant le traitement, l'instruction à répéter peut ne pas être exécutée du tout
  - **Nombre de répétitions connu a priori** : boucle avec compteur
    `for`

Les boucles

La boucle `for`

- **Nombre d'itérations connu a priori**
- Boucle avec compteur : on indique
  - la valeur de départ
  - la valeur de fin
  - le pas d'incrémentation à la fin de chaque itération

```
for uneVar in range(debut, fin, incrementation ) :
  # instruction(s) a repeter
```

- Remarques :
  - `uneVar` prendra toutes les valeurs depuis `debut` jusqu'à `fin` (exclus) par pas de `incrementation`
  - la valeur de début et l'incrémentation ne sont pas obligatoires :

```
for uneVar in range(fin) :
  # instruction(s) a repeter
```

    On va alors de 0 à `fin-1`
  - en Python l'indentation est **cruciale**

Les boucles : Exemples

Parcours classique

```
# Affichage des entiers de 0 a 2
for i in range(3) :
  print( i )
```

Parcours inverse

```
# Affichage des entiers de 3 a 1
for i in range( 3, 0, -1 ) :  # Decrementation
  print( i )
```

Incrémentation quelconque

```
for i in range(0, 10, 2 ) :  # On incremente i de 2
  print(i)  # Affiche les entiers pairs
```

SCIENCES SORBONNE UNIVERSITÉ

La boucle `while`

Principe

On ne connaît pas toujours le nombre d'opérations nécessaires pour obtenir un résultat

- données saisies par un utilisateur (numéro de téléphone, *etc.*) avec risque d'erreur de saisie (pas que des chiffres, pas la bonne longueur, *etc.*)
- calcul itératif de la limite d'une suite mathématique, du zéro d'une fonction avec une borne sur l'erreur
- recherche d'une valeur particulière dans un ensemble de données

La condition de continuation ne porte plus (uniquement) sur un compteur

SCIENCES SORBONNE UNIVERSITÉ

La boucle `while`

Définition

La condition de continuation est évaluée **au début** de la boucle

- elle porte sur une ou plusieurs variables initialisées avant la boucle
- si la condition est fausse initialement, le programme n'exécute aucune instruction de la boucle
- si la condition de continuation ne devient jamais fausse, le programme ne sort jamais de la boucle (on parle de boucle infinie)
  $\Rightarrow$ dans la boucle, une ou plusieurs instructions agissent sur les variables de la condition de continuation pour la faire évoluer vers la condition d'arrêt

Syntaxe

```
while condition :
  # instruction(s) a repeter
```

SCIENCES SORBONNE UNIVERSITÉ

Les boucles : Exemples

Affichage de tous les entiers positifs inférieurs à 10

On connaît

- la valeur de départ (1)
- la valeur de fin (10)
- l'incrémentation (+1)

$\Rightarrow$ on peut utiliser une boucle `for`

```
for i in range(1, 11) :   # Incrementation = 1 donc optionnelle
  print( i )              # Affiche tous les entiers de 1 a 10
```

Les boucles : Exemples

Simulation de la division entière

On calculera le quotient et le reste de la division entière de a par b (entrés par l'utilisateur) sans utiliser les opérateurs // et %.

- On connaît les valeurs de départ ($q = 0$, $r = a$)
- On ne connaît pas le nombre d'itérations à faire (valeur du quotient)
- On peut ne pas passer dans la boucle (si $b > a$)

$\Rightarrow$ on peut utiliser une boucle `while`

```
# On demande a et b a l'utilisateur
a = int( input( ) )
b = int( input( ) )
q, r = 0, a
while r >= b :
  q += 1    # On incremente le quotient
  r -= b    # On retranche b au reste => modification d'une
            # variable dans la condition d'arret
print( a, " = ", b, " * ", q, " + ", r )
```

Les boucles : Exemples

Somme d'entiers

On demandera à l'utilisateur d'entrer des entiers dont on calculera la somme. On s'arrêtera si la valeur entrée est négative.

- On ne connaît pas le nombre de valeurs que l'utilisateur va donner
- On demandera au moins une valeur à l'utilisateur

$\Rightarrow$ on peut utiliser une boucle `while`

```
i = int( input( ) )     # Initialisation
somme = 0
while i > 0 :
  i = int( input( ) )   # On modifie la variable de la
  if i > 0 :            # condition de continuation =>
    somme += i          # Pas de boucle infinie
print( somme )
```

SCIENCES SORBONNE UNIVERSITÉ

Les boucles : Exemple

Conversion boucle `for` vers `while`

```
# Initialisation, test et incrementation dans le for
for i in range( 0 , 4 , 1 ) :
  print( i )
```

```
i = 0                # Initialisation
while i < 4 :        # Test
  print( i )
  i += 1             # Incrementation
```

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Exemple introductif

Signe d'un entier

```
a = int( input( ) )
if a > 0 : print( "a =", a, "est positif" )
else :     print( "a =", a, "est negatif ou nul" )

...   # Des instructions modifiant a

if a > 0 : print( "a =", a, "est positif" )
else :     print( "a =", a, "est negatif ou nul" )

b = int( input( ) )  # Une nouvelle variable
if b > 0 : print( "b =", b, "est positif" )
else :     print( "b =", b, "est negatif ou nul" )
```

Problème

- Duplication / répétition de code ⇒ Code plus long (donc moins lisible)
- Modification de code :
  - nouvelle fonctionnalité (*e.g.* afficher un message si l'entier est nul)
  - correction d'erreur(s)
- Lisibilité : que faire quand le programme fait 1000 lignes ? 1 000 000 lignes ?

Exemple introductif

Signe d'un entier

```
a = int( input( ) )
if a > 0 : print( "a =", a, "est positif" )
else :     print( "a =", a, "est negatif ou nul" )

...   # Des instructions modifiant a

if a > 0 : print( "a =", a, "est positif" )
else :     print( "a =", a, "est negatif ou nul" )

b = int( input( ) )  # Une nouvelle variable
if b > 0 : print( "b =", b, "est positif" )
else :     print( "b =", b, "est negatif ou nul" )
```

Solution

On voudrait pouvoir écrire

```
afficher le signe de a
```

Fonctions

Qu'est ce qu'une fonction ?

- Une fonction définit le programme permettant de résoudre un problème.
- Ce problème peut lui-même être décomposé (appel à d'autres fonctions).

$\Rightarrow$ un des mécanismes de base en programmation

Pourquoi écrire des fonctions ?

- Factorisation de code répétitif $\Rightarrow$ moins de travail pour le programmeur
- Amélioration de la lisibilité $\Rightarrow$ donne un nom à du code
- Découpage fonctionnel du programme $\Rightarrow$ décompose un problème en sous-problèmes
- Correction d'erreurs plus facile $\Rightarrow$ on peut tester chaque fonction séparément, 1 seul endroit à modifier
- Ré-utilisabilité / Mutualisation de code $\Rightarrow$ on peut réutiliser des fonctions dans d'autres projets ou les partager avec d'autres programmeurs.

Déclaration

Syntaxe

Mot clé : `def`

```
def nomMethode( parametres ) :
  # Corps de la methode
  # Attention a l'indentation!!
```

- `parametres` : liste des variables (ou paramètres) nécessaires à la fonction. S'il y a plusieurs paramètres, ils sont séparés par une virgule

Exemples

```
def afficher( a ) :
  print( "a =", a )

def somme( a, b ) :
  return a + b
```

SCIENCES SORBONNE UNIVERSITÉ

Paramètre(s) d'une fonction

Syntaxe

```
def nomMethode( parametres ) :
  # Corps de la methode
```

Principe

On donne dans la parenthèse après le nom de la fonction :

- la liste des paramètres (leurs noms) de la fonction
- les paramètres sont séparés par une virgule

Remarque : si la fonction n'a aucun paramètre $\Rightarrow$ parenthèse vide

SCIENCES SORBONNE UNIVERSITÉ

Paramètre(s) d'une fonction

Syntaxe

```
def nomMethode( parametres ) :
  # Corps de la methode
```

Exemples

```
# Fonction sans parametre
def afficheLoutre( ) :
  print( "Vive les loutres!" )

# Fonction avec un parametre
def afficheSigne( n ) :
  if n >= 0 :
    print( "n est positif" )
  else :
    print( "n est negatif" )

# Fonction avec 2 parametres
def afficheSigne( n, nom ){
  print( nom, "est", "positif" if n >= 0 else "negatif" )
```

SCIENCES SORBONNE UNIVERSITÉ

Paramètre(s) d'une fonction

Syntaxe

```
def nomMethode( parametres ) :
  # Corps de la methode
```

Exemples

```
# Fonction avec 2 parametres
def afficheMax( a, b ) :
  if a >= b :
    print( "Le max est", a )
  else
    print( "Le max est", b )

# Fonction avec 3 parametres
def afficheMax( a, b, c ) :
  m = a
  m = b if m < b
  m = c if m < c
  print( "Le max est", m )
```

SCIENCES SORBONNE UNIVERSITÉ

Valeur de retour

Syntaxe

```
def nomMethode( parametres ) :
  # Corps de la methode
  ...
  return uneValeur   # Resultat de la fonction
```

- Valeur de retour = résultat de la fonction
- Mot clé `return`
  - arrête l'exécution de la fonction
  - renvoie la valeur qui suit
- Pas de valeur de retour ⇒
  - l'exécution de la fonction s'arrête à la fin du bloc
  - Pas de `return`

SCIENCES SORBONNE UNIVERSITÉ

Valeur de retour

Syntaxe

```
def nomMethode( parametres ) :
  # Corps de la methode
  ...
  return uneValeur   # Resultat de la fonction
```

Exemples

```
def somme( a, b ) :
  res = a + b
  return res

def max( a, b ) :
  if a >= b :
    return a     # Ok: valeur de retour
  else :
    return b     # connue pour les 2 cas

def affiche( x ) :
  print( "x =", x )
  # Ok: valeur de retour => pas de return
```

Déclaration de fonction

Quelques erreurs (syntaxiques)

```
def somme( a, b ) :
  res = a + b      # oubli du return
                   # le resultat est perdu

somme( a, b ) :    # oubli du mot cle def
  return a + b

def max( a, b ) :
  if a >= b :
    return a       # oubli du return si a < b

def affiche( x ) :
  printf( "x =", x )
    return 1       # Mauvaise indentation

def somme( a, b ) :
  return a + b
  # la ligne suivante ne sera jamais executee
  print( "Le resultat est", (a+b) )
```

SCIENCES SORBONNE UNIVERSITÉ

Écriture de fonctions

Conseils

- Commencer par se demander de quoi la fonction a besoin (paramètres) et si elle renvoie un résultat (de quel type) ou pas
- Différencier les fonctions qui réalisent des affichages et les fonctions qui retournent des résultats
  $\Rightarrow$ une fonction qui produit un résultat n'affiche généralement rien !

[Alan Perlis]

"Si vous avez une fonction avec 10 paramètres, vous en avez probablement oublié."

SCIENCES SORBONNE UNIVERSITÉ

Écriture de fonctions

Bonnes pratiques

- Regarder s'il n'existe pas déjà une fonction qui résout le problème
- Écrire le plus de fonctions possible et les plus petites possible
- Mettre un commentaire au dessus pour expliquer ce que fait la fonction et comment elle s'utilise.

SCIENCES SORBONNE UNIVERSITÉ

Appel d'une fonction

Principe

- La définition d'une fonction n'exécute pas le programme (suite d'instructions) qui la compose. Il faut l'appeler depuis le programme principal ou une autre fonction.
- Paramètres formels, paramètres effectifs :
  - Les paramètres formels sont utilisés lors de la définition de la fonction.
  - Lors de l'appel, des valeurs doivent être données à ces paramètres → paramètres effectifs.
  - Un paramètre effectif est soit une variable (sa valeur), soit une valeur d'expression, soit une valeur de retour d'un appel de fonction.

Syntaxe

- Fonction sans valeur de retour :

```
nomMethode( paramètres_effectifs )
```

- Fonction avec valeur de retour :

```
variable = nomMethode( paramètres_effectifs )
```

SCIENCES SORBONNE UNIVERSITÉ

Appel d'une fonction

Remarques

- La méthode doit toujours être déclarée avant d'être appelée (sinon le programme ne la connait pas !)
- Le nombre et l'ordre des paramètres formels et des paramètres effectifs d'une fonction doivent être identiques.
- Les variables utilisées au sein d'une fonction doivent être déclarées comme variables locales de la fonction ou doivent correspondre à des paramètres de la fonction.

Appel d'une fonction

Exemple

```
def somme( a, b ) :
    return a + b

def afficheSomme( a, b ) :
    s = somme( b, a )  # Appel avec 2 variables
    print( "La somme est", s )

x = 4
# Appel avec 1 variable et 1 expression
res = somme( x, 38 )
print( "res =", res )
# Appel avec 1 expression et 1 valeur de retour de fonction
afficherSomme( 13, somme( x, 38 ) )
```

- ?? : paramètres formels de la méthode
- ?? : paramètres effectifs de la méthode

SCIENCES SORBONNE UNIVERSITÉ

Appel d'une fonction

Quelques erreurs (syntaxiques)

```
res = somme( 4, 2 )  # Appel avant la declaration

def somme( a, b ) :
  return a + b

def droite( a, b ) :
  res = a * x + b    # x n'est pas connu
  return res

x = 4
d = 8.
res = somme( x )  # Mauvais nombre de parametres
```

SCIENCES SORBONNE UNIVERSITÉ

Portée des variables

Variables locales

- Une variable déclarée à l'intérieur d'une fonction est une variable locale.

Portée des variables

- La portée d'une variable désigne la partie du programme dans laquelle on peut l'utiliser.
- Une variable locale n'est utilisable qu'à l'intérieur de la fonction dans laquelle elle est déclarée.
- Les paramètres formels d'une fonction correspondent à des variables locales initialisées lors de l'appel de la fonction.

Portée des variables

Exemple

```
def somme( a, b ) :
    res = a + b
    return res

a = 4
print( somme( 38, a ) )
```

- a, b : portée des paramètres de la fonction `somme`
- res : portée de la variable locale de la fonction `somme`
- a : portée de la variable locale de la fonction `main`

Remarques

- Le nom d'une variable passée à une fonction n'a pas d'importance, seule sa valeur est transmise (*e.g.* dans le programme principal, a = 4 et dans `somme`, a = 38).

SCIENCES SORBONNE UNIVERSITÉ

Portée des variables

Quelques erreurs (syntaxiques)

```
def somme( a, b ) :
  x = a + n  # n n'est pas defini dans somme
  return a + b

n = 4
x = somme( n, 38 )
print( a, ",", b )  # a et b pas definis ici
```

Quelques fonctions utiles

Fonctions prédéfinies

- `round( x )` : arrondi au plus proche de x
- `abs( x )` : calcul de $|x|$

Le module `math`

- Définition de nombreuses fonctions (et constantes) mathématiques dont :
  - `sqrt( x )` : $\sqrt{x}$
  - `cos( x )`, `sin( x )`, `tan( x )` : $\cos$, $\sin$, $\tan$ (avec x en radians)
  - `exp( x )`, `log( x )` : e^x et $\ln(x)$
  - `math.pi` : π

Le module `random`

- Définition de générateurs de nombres aléatoires dont :
  - `random( )` : réel aléatoire (uniforme) dans $[0; 1[$
  - `randint( a, b )` : entier aléatoire (uniforme) dans $[a; b]$

SCIENCES SORBONNE UNIVERSITÉ

Quelques fonctions utiles

Exemples

```
# Inclusion des modules
import math
import random

# Arrondi et valeur absolue
x = abs( round( 13.37 ) )

# Calcul de cos( pi / 4 )
cosPi4 = math.cos( math.pi / 4 );
# Comparaison de cos( pi / 4 ) avec 1/sqrt(2)
cosEgal = (cosPi4 == (1 / math.sqrt(2) ) )

# Lancer d'un de a 6 faces
de = random.randint(1, 6)
```

SCIENCES SORBONNE UNIVERSITÉ

Plan

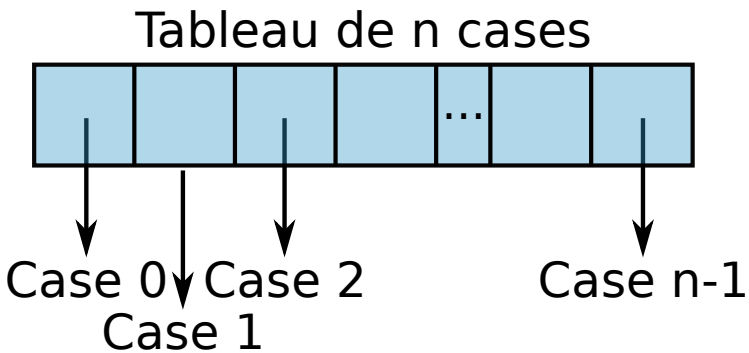
SCIENCES SORBONNE UNIVERSITÉ

Qu'est ce qu'un tableau

Définition

Ensemble

- de taille fixe
- de variables du même type
- adressé par un indice (ou numéro) : la position / case dans le tableau
- contigüe en mémoire



SCIENCES SORBONNE UNIVERSITÉ

Qu'est ce qu'un tableau

Définition

Ensemble

- de taille fixe
- de variables du même type
- adressé par un indice (ou numéro) : la position / case dans le tableau
- contigüe en mémoire

Exemple

- Résultats de plusieurs lancers de "Pile ou Face" : tableau de booléens
- Numéros d'étudiants d'un groupe de TD : tableau d'entiers
- Coordonnées d'un vecteur : tableau de réels
- Plaque d'immatriculation des véhicules d'une entreprise : tableau de chaînes de caractères

SCIENCES SORBONNE UNIVERSITÉ

Déclaration d'un tableau

Syntaxe

- Tableau vide (aucune case) : `nomTab = [ ]`
- Tableau de n cases remplies de valeurs identiques : `nomTab = [ val ] * n`
- Tableau avec des valeurs différentes : `nomTab = [ val1, val2, val3, ... ]`

Exemple

```
tabEntier = [ ]                 # Un tableau vide
tabEntier1 = [ 0 ] * 4          # Tableau de 4 cases remplies de 0
# Un tableau de 4 entiers differents
tabEntier2 = [ 1, 3, 3, 7 ]   # Son contenu est 1, 3, 3, 7
```

SCIENCES SORBONNE UNIVERSITÉ

Utilisation d'un tableau

Utilisation d'un tableau

- Accès à une valeur
  - On accède au contenu d'une case d'un tableau par `tab[i]`
  - Attention : les indices (i) vont de 0 à n-1 (où n est la taille du tableau)
  - L'indice -1 permet de récupérer la dernière case du tableau
- Ajout d'une valeur à la fin du tableau : `tab.append( val )`
- Longueur d'un tableau : `len( tab )`

Exemple

```
tabEntier = [1, 3, 3, 7]
tabEntier[2] = 42       # Modification de la 3eme case du tableau
i = tabEntier[0]        # Copie de la 1ere case dans la variable i

tabEntier.append(142857)  # Ajout d'une valeur a la fin (5 cases)
print( tabEntier[-1] )    # Affiche la derniere case du tableau

lTab = len( tab )       # Longueur du tableau
tabEntier[lTab] = 2     # Erreur!
```

SCIENCES SORBONNE UNIVERSITÉ

Utilisation d'un tableau

Sous-ensemble d'un tableau

- Syntaxe "Matlab" : `sousTab = tab[a:b:c]`
  - a : indice de départ (0 par défaut)
  - b : indice de fin (dernier élément par défaut)
  - c : pas (1 par défaut)
- Remarque : on récupère les cases de $[a; b[$

Exemple

```
tab = [i for i in range(20)]  # Tous les entiers de 0 a 19

sTab = tab[0:10:1]    # Toutes les cases de 0 a 10 (exclu)
sTab = tab[0:10:2]    # Cases de 0 a 10 (exclu) par pas de 2

# "Parcours inverse"
sTab = tab[10:0:-1]   # Cases de 10 a 0 (exclu)

# Certains parametres peuvent etre omis
sTab = tab[0:10]      # Toutes les cases de 0 a 10 (exclu)
sTab = tab[:10:2]     # Cases de 0 a 10 (exclu) par pas de 2
sTab = tab[10::]      # Cases de 10 a la fin par pas de 1
sTab = tab[::2]       # Cases d'indice pair (0, 2, 4, ...)
```

SCIENCES SORBONNE UNIVERSITÉ

Parcours complet d'un tableau

Parcours de toutes les valeurs d'un tableau

Il faut accéder successivement au contenu de chaque case du tableau

$\Rightarrow$ Utilisation d'une boucle `for`

```
# Boucle de la 1ere case (0) a la derniere (len(tab)-1)
for i in range( len(tab) ) :
  # Serie d'instructions
```

Exemples

- Affichage des valeurs
- Copie / égalité des valeurs du tableau
- Recherche de la valeur minimum / maximum
- Calcul de la somme / moyenne des valeurs

SCIENCES SORBONNE UNIVERSITÉ

Parcours complet d'un tableau

Parcours de toutes les valeurs d'un tableau

Il faut accéder successivement au contenu de chaque case du tableau

⇒ Utilisation d'une boucle `for`

```
# Boucle de la 1ere case (0) a la derniere (len(tab)-1)
for i in range( len(tab) ) :
  # Serie d'instructions
```

Exemples

```
# Affichage de toutes les valeurs
for i in range( len(tab) ) :
  print( tab[i], " ", end="" )
print( "" )  # Retour a la ligne

# Recherche du maximum et de sa position
maxT, posT = 0
for i in range( len(tab) ) :
  if maxT < tab[i] :
    maxT = tab[i]
    posT = i
```

SCIENCES SORBONNE UNIVERSITÉ

Parcours complet d'un tableau

Parcours complet : syntaxes abregées

Si on a besoin

- de la valeur des cases du tableau : `for valT in tab`
- de la valeur et des indices des cases du tableau : `for i, valT in enumerate(tab)`
  `enumerate` renvoie successivement tous les couples (indice, valeur) d'un tableau

Exemples

```
# Affichage de toutes les valeurs
for valT in tab :  # valT prend successivement toutes
  print( valT, " ", end="" )        # les valeurs de tab
print( "" )  # Retour a la ligne

# Recherche du maximum et de sa position
maxT, posT = tab[0], 0
# enumerate renvoie successivement tous les indices
for i, valT in enumerate( tab ) :  # et valeurs de tab
  if maxT < valT :
    maxT = valT
    posT = i
```

SCIENCES SORBONNE UNIVERSITÉ

Parcours partiel d'un tableau

Parcours partiel des valeurs d'un tableau

Il faut accéder successivement au contenu de chaque case du tableau sans connaître le nombre de répétitions

$\Rightarrow$ Utilisation d'une boucle `while`

```
i = 0
while i < len(tab) and condition d arret :
  # Serie d instructions
  ...
  i += 1  # Incrementation
```

Exemples

- Recherche d'une valeur dans le tableau
- Parcours d'un tableau surdimensionné

SCIENCES SORBONNE UNIVERSITÉ

Parcours partiel d'un tableau

Parcours partiel des valeurs d'un tableau

Il faut accéder successivement au contenu de chaque case du tableau sans connaître le nombre de répétitions

$\Rightarrow$ Utilisation d'une boucle `while`

```
i = 0
while i < len(tab) and condition d arret :
  # Serie d instructions
  ...
  i += 1  # Incrementation
```

Exemple : Recherche du premier 5 dans un tableau d'entiers

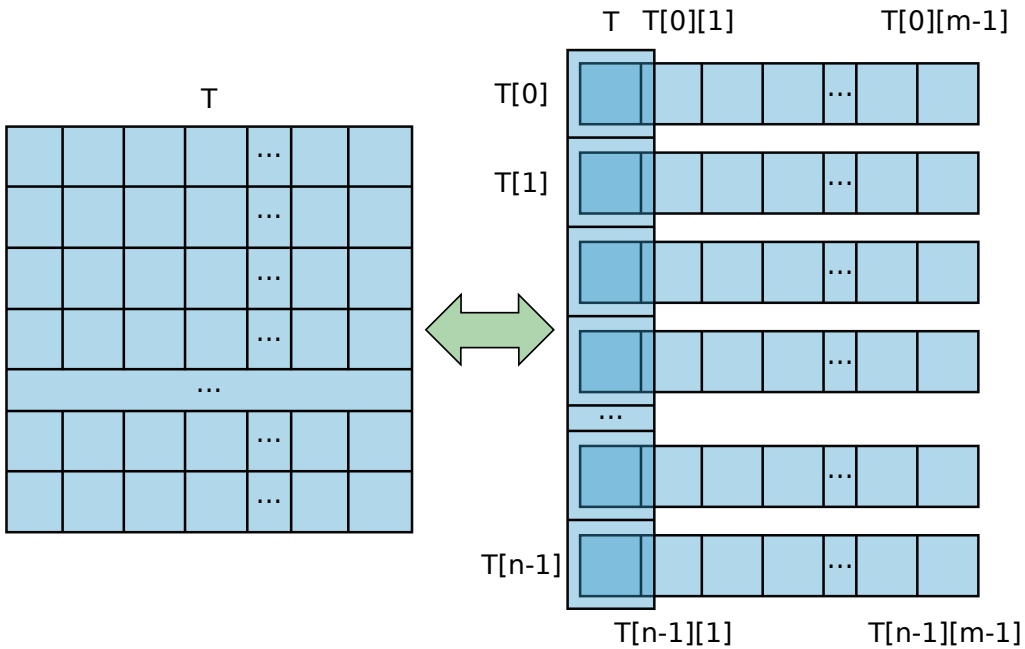
```
i = 0
while i < len(tab) and tab[i] != 5 :
  i += 1
if i == len(tab) :
  print( "Pas de 5 dans le tableau" )
else
  print( "Le 1er 5 est a la position", i )
```

SCIENCES SORBONNE UNIVERSITÉ

Matrice

Définition

- Un tableau 2D (ou matrice) peut être vu comme un tableau contenant des tableaux contenant des éléments
- Déclaration
  - Matrice vide (aucune case) : `mat = [ [ ] ]`
  - Matrice de $n \times m$ cases remplies de valeurs identiques : `mat = [ [val] * m ] * n`
  - Matrice avec des valeurs différentes :
    `mat = [ [val11, val12, val13, ...], [val21, val22, val23, ...], ... ]`
- Utilisation : accès à la j$^{\text{ème}}$ case de la i$^{\text{ème}}$ ligne : `mat[i][j]`



Exemple

Parcours de toutes les valeurs d'une matrice

```
from random import randint
# Matrice (n x m) d'entiers aleatoires entre a et b
n, m = 13, 37
a, b = 1, 6
mat = [ [ randint(a, b) for j in range(m) ] for i in range(n) ]

# Recuperation d'une ligne
tab = mat[1]

# Affichage de toutes les valeurs
  # Parcours des lignes
for i in range( len(mat) ) :
  # Parcours des colonnes
  for j in range( len(mat[i]) ) :
    print( mat[i][j], " ", end="" )
  print( "" )
```

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Chaînes de caractères

Définitions

- Caractères ou suite de caractères
- Tous les caractères de la table ASCII sont autorisés (*i.e.* lettres, chiffres, symboles)

Syntaxe

- Valeur donnée entre guillemets " ou entre apostrophe '
- Opérateur "+" :
  - Permet de concaténer plusieurs chaines de caractères
  - Si l'un des opérandes n'est pas une chaine de caractères, il faut d'abord le convertir avec `str(val)`
- **Jamais de caractères accentués !**
- **Remarque :** Une chaine de caractères est un tableau
  - Longueur de la chaîne : `len(chaine)`
  - Accès à un caractère : `c = chaine[pos]`
    `pos` est un entier variant de 0 à `len(chaine)-1`

Comparaison de chaînes

- On peut comparer 2 chaînes de caractères avec les opérateurs `==`, `<`, `>`

SCIENCES SORBONNE UNIVERSITÉ

Chaînes de caractères

Exemples

```
# Declaration de chaines de caracteres
chaine = 'Vive les loutres'
chaine2 = "Des nombres (42, 142.857) et des symboles (&, _)"

# Concatenation de 2 chaines
chaine += " et Charlie"
# chaine vaut "Vive les loutres et Charlie"

# Concatenation d'une chaine et d'un nombre
chaine = "La reponse est " + str(42)
# chaine vaut "La reponse est 42"

# Concatenation de chaines et de variables
i = 4
x = 3.
chaine = str(x) + "^" + str(i) + " = " + str(x**i)
# chaine vaut "3.0^4 = 81.0"

# Comparaison de chaines
print( "Charlie" < "Loutre" )    # affiche True
```

Fonctions importantes

Remarques

- Attention à la manière d'appeler ces méthodes :
  `nomVariable . nomMethode ( parametres )`

Recherche d'un caractère dans la chaîne

- `idx = chaine.find( car )`
- Depuis une position : `idx = chaine.find( car, debut, fin )`
- Depuis la fin : `idx = chaine.rfind( car )`
- Depuis la fin et à partir d'une position : `idx = chaine.rfind( car, debut, fin )`

Remarque : toutes ces fonctions renvoient la position de la première (ou dernière) apparition du caractère car et -1 s'il n'est pas trouvé

SCIENCES SORBONNE UNIVERSITÉ

Fonctions importantes

Remarques

- Attention à la manière d'appeler ces méthodes :
  `nomVariable . nomMethode ( parametres )`

Manipulation de la chaîne

- Extraction d'une sous-chaîne : `chaine2 = chaine[deb:fin]`
  Remarque : le caractère de position `fin` est exclu
- Séparation de la chaîne : `tabChaine = chaine.split( c )`
  Remarque : `split` scinde `chaine` à chaque occurence du caractère `c` et renvoie un tableau contenant toutes les sous-chaînes ainsi trouvées (le caractère `c` est supprimé)

Manipulation des caractères

- Remplacement de caractères : `chaine2 = chaine.replace( oldChar, newChar )`
- Conversion de la chaîne en minuscule : `chaine2 = chaine.lower( )`
- Conversion de la chaîne en majuscule : `chaine2 = chaine.upper( )`

Exemples

Exemples

```
chaine = "Vive les loutres"
print( "Longueur de chaine", len(chaine) )
cE = chaine[3]  # cE = 4eme caractere de chaine:'e'

posE = chaine.find( cE )
if posE == -1 :  print( "La chaine ne contient pas de " + cE )
else :           print( "Le premier", cE, "est en", posE )
print( "Le dernier", cE, "est en", chaine.rfind( cE ) )

chaine2 = chaine[0:5] + "Charlie"
    # chaine2 = "Vive Charlie"
chaine2 = chaine.replace( 'e', 'a' )
    # chaine2 = "Viva las loutras"
tabChaine = chaine.split( ' ' )
    # tabChaine = [ "Vive", "les", "loutres!" ]

egal = chaine == chaine2  # chaine = chaine2? => False

chaine3 = "CHaiNe QuELconqUE & 12!";
chaine3 = chaine3.lower()  # chaine3 = "chaine quelconque & 12!"
```

Tableaux et chaînes de caractères

Quelques erreurs (syntaxiques)

```
tab = [1, 3, 3, 7]
i = tab[4]  # Derniere case du tableau: 3
tab[4] = 3  # Ajout avec append
append( tab, 3 )  # Mauvais appel de la fonction

chaine = "Vive les loutres"
chaine[2] = "b"   # Les chaines ne peuvent pas etre modifiees
idx = find( chaine, "e" )  # Mauvais appel de la fonction
```

SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

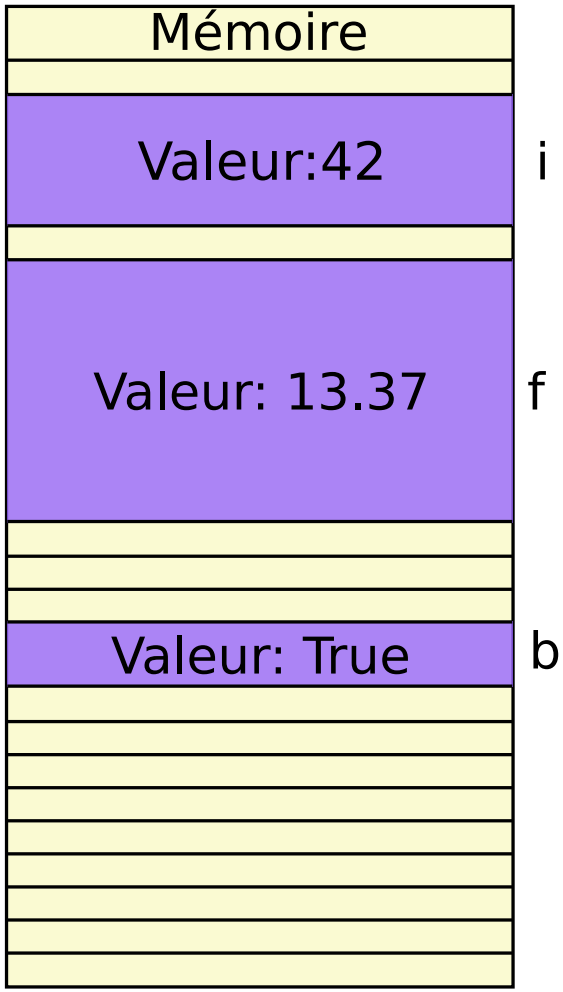
Représentation mémoire des variables

Type "primitif" (entier, réel, booléen)

- La variable stocke directement la valeur

Exemple

```
i = 42
b = True
f = 13.37
```



SCIENCES SORBONNE UNIVERSITÉ

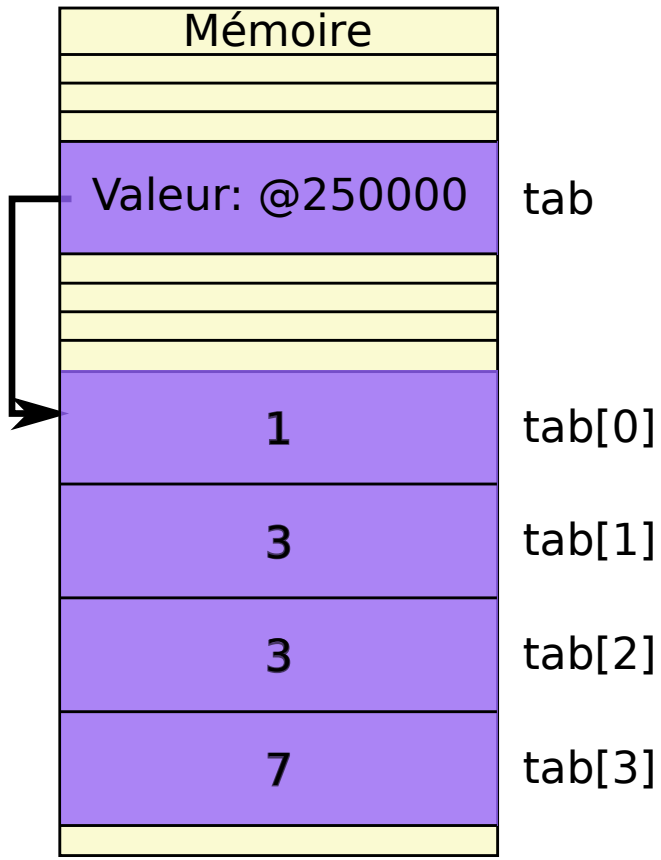
Représentation mémoire des variables

Tableau et chaîne de caractères

- La variable stocke l'adresse à laquelle la valeur est conservée (de manière contigüe)
- **Remarque :** une variable de type non primitif est donc un pointeur

Exemple

```
tab = [1, 3, 3, 7]
```



SCIENCES SORBONNE UNIVERSITÉ

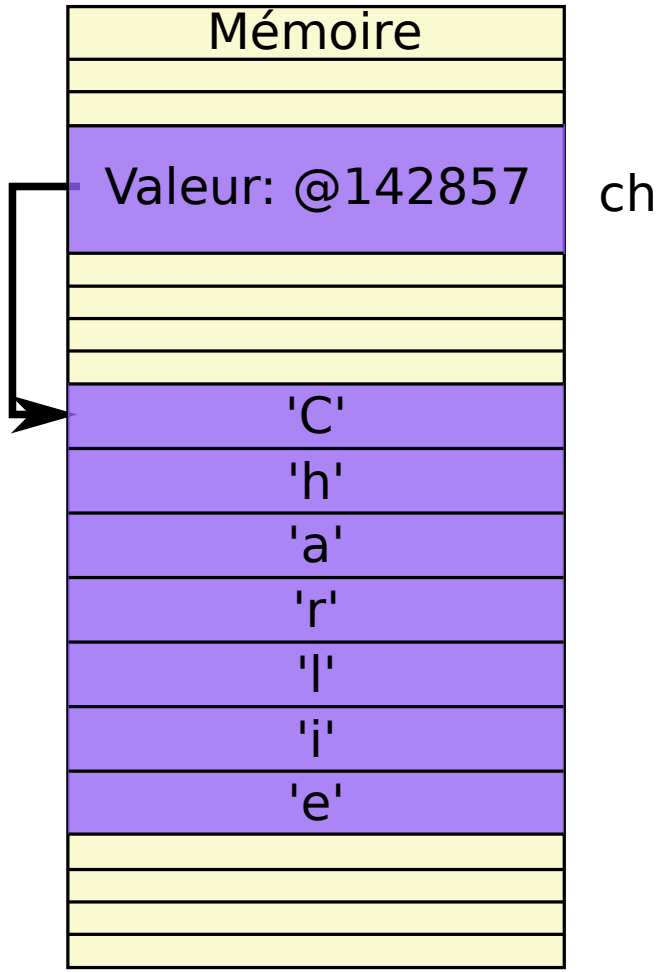
Représentation mémoire des variables

Tableau et chaîne de caractères

- La variable stocke l'adresse à laquelle la valeur est conservée (de manière contigüe)
- **Remarque :** une variable de type non primitif est donc un pointeur

Exemple

```
ch = "Charlie"
```



SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par valeur / copie

- La valeur de la variable est copiée à un autre endroit de la mémoire (que la fonction appelante ne connaît pas)
- À la fin de la fonction, la copie est détruite
- Les types primitifs (`int`, `float`, `bool`) sont gérés par copie

Conséquences

- Les modifications apportées à une variable passée par copie ne sont donc pas connues de la fonction appelante

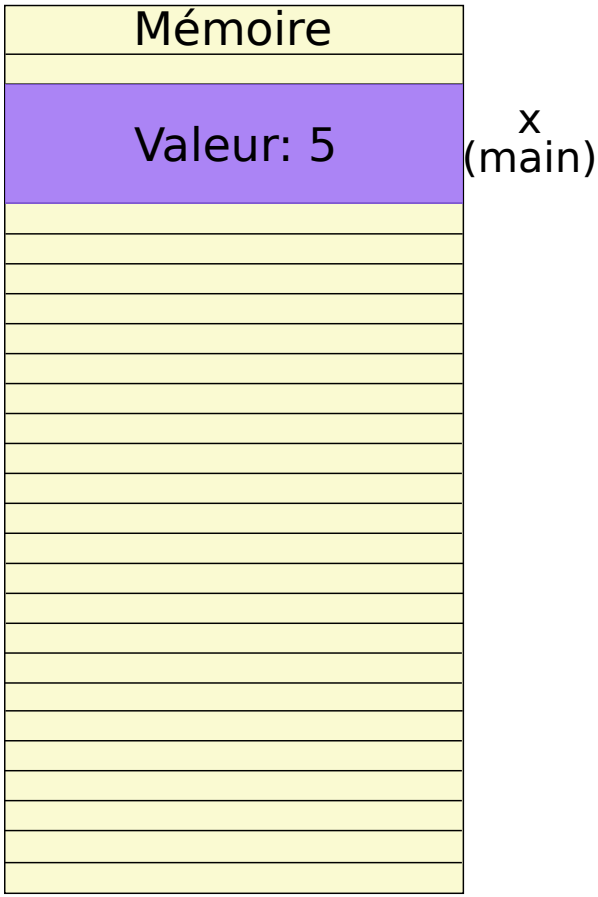
SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par valeur

```
def modifX( x ) :
  x = 9

x = 5
modifX( x )
print( x )
```



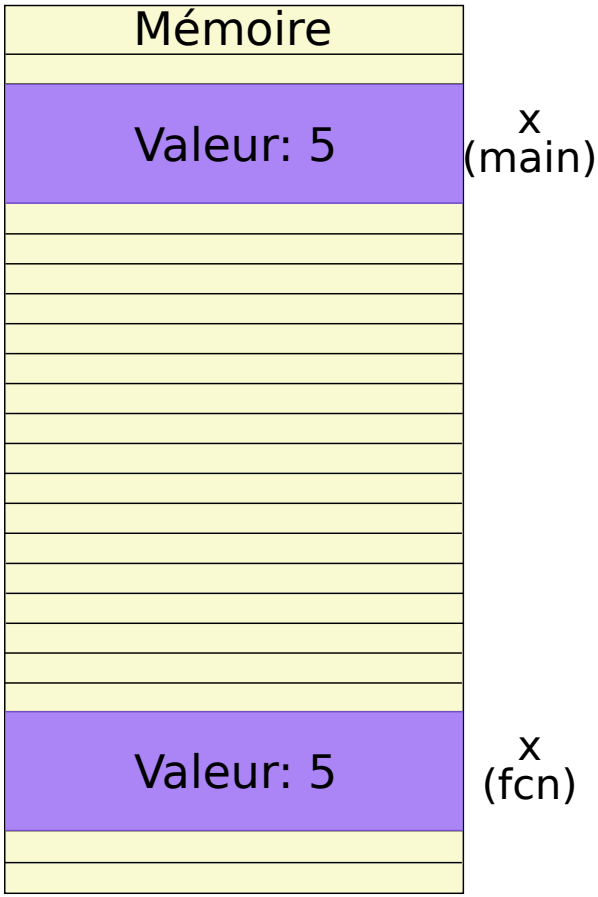
SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par valeur

```
def modifX( x ) :
  x = 9

x = 5
modifX( x )
print( x )
```



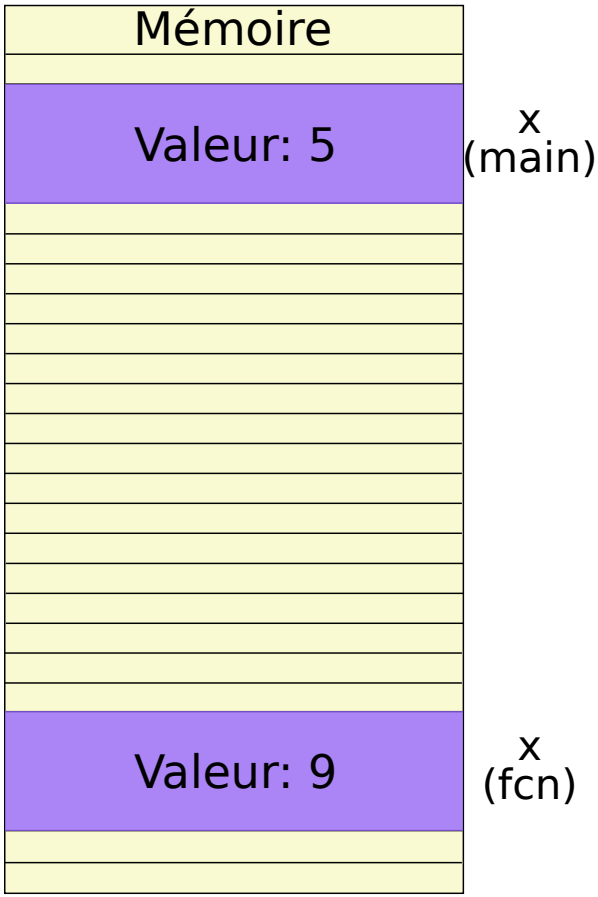
SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par valeur

```
def modifX( x ) :
    x = 9

x = 5
modifX( x );
print( x )
```



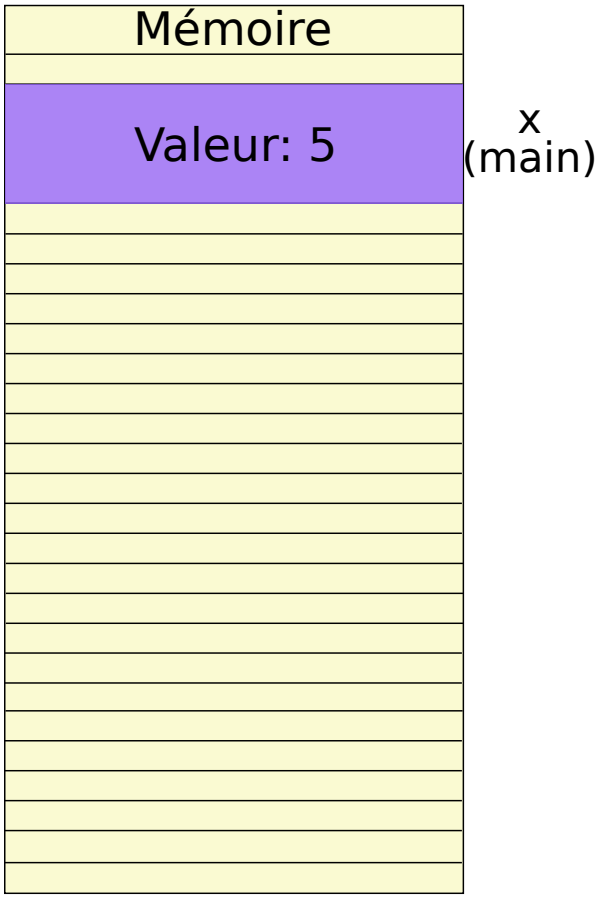
SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par valeur

```
def modifX( x ) :
  x = 9

x = 5
modifX( x )
print( x )
```



SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par adresse

- L'adresse de la variable est copiée à un autre endroit de la mémoire (que la fonction appelante ne connaît pas)
- À la fin de la fonction, la copie de l'adresse est détruite (mais pas la partie pointée)
- Les types non-primitifs sont gérés par adresse

Conséquences

- Les modifications apportées à la variable seront visibles dans la fonction appelante
- Si on veut appliquer un traitement à un tableau ou une chaîne de caractères tout en gardant l'original, il faut donc commencer par copier chaque élément de la variable

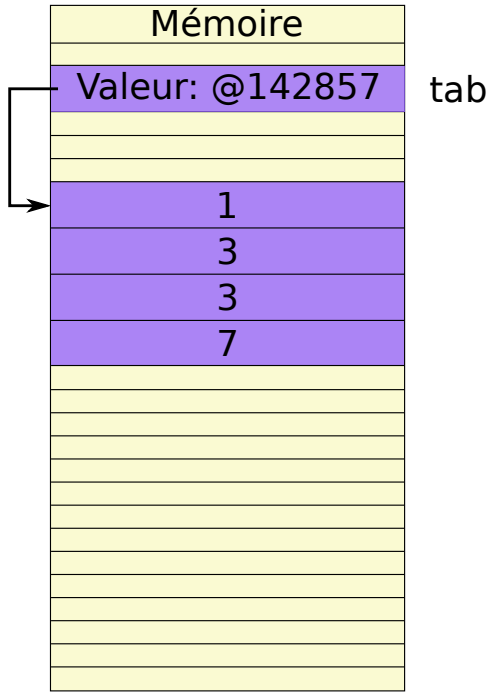
SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par adresse

```
def modifTab( tabF ) :
  tabF[3] = 42

tab = [1, 3, 3, 7]
modifTab( tab )
for i in tab :
  print( i )
```



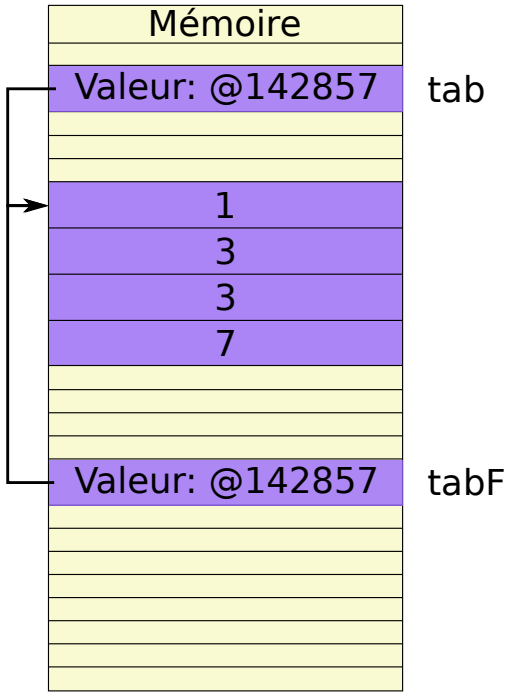
SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par adresse

```
def modifTab( tabF ):
  tabF[3] = 42

tab = [1, 3, 3, 7]
modifTab( tab )
for i in tab :
  print( i )
```



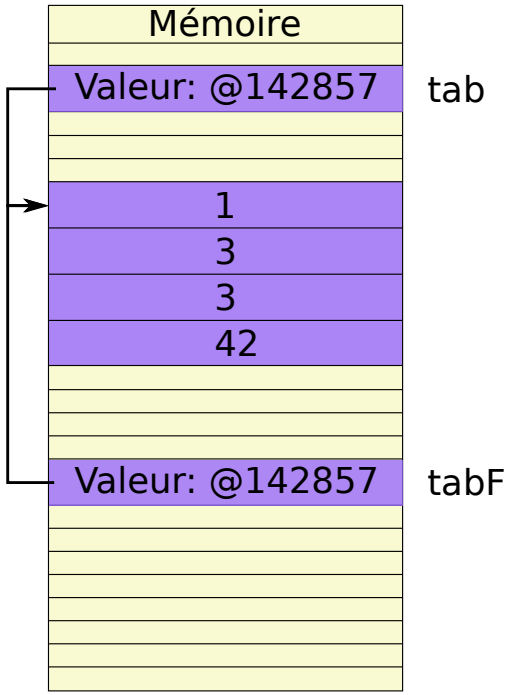
SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par adresse

```
def modifTab( tabF ) :
    tabF[3] = 42

tab = [1, 3, 3, 7]
modifTab( tab )
for i in tab :
    print( i )
```



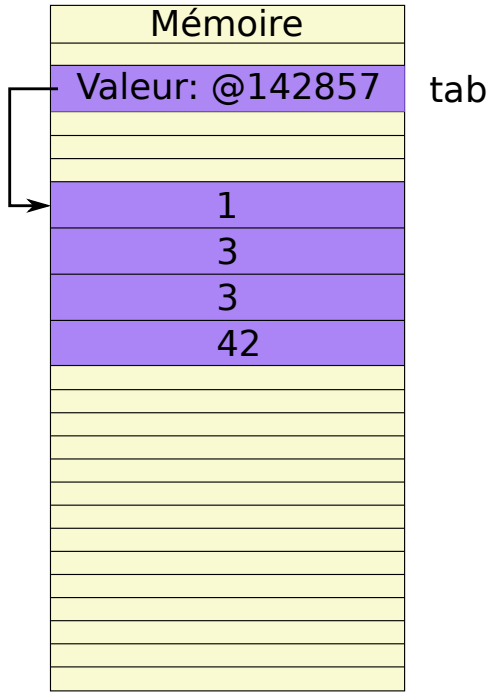
SCIENCES SORBONNE UNIVERSITÉ

Passage de paramètres

Passage par adresse

```
def modifTab( tabF ) :
  tabF[3] = 42

tab = [1, 3, 3, 7]
modifTab( tab )
for i in tab :
  print( i )
```



SCIENCES SORBONNE UNIVERSITÉ

Plan

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Syntaxe de Python

Le Rubik's cube

Un Rubik's Cube est un cube composé de plusieurs mini-cubes qui pivotent autour du centre selon 3 axes.

Question

- Écrire un programme qui affiche le nombre de mini-cubes visibles sur un cube de taille N donnée.

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Syntaxe de Python / Fonction

Calcul de $\sin(x)$

On souhaite calculer les valeurs de $\sin(x)$ avec une précision ϵ. On rappelle que

$$\sin(x) = \sum_{n=0}^{+\infty} \left(\frac{(-1)^n x^{2n+1}}{(2n+1)!} \right)$$

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Syntaxe de Python / Fonction

Calcul de $\sin(x)$

On souhaite calculer les valeurs de $\sin(x)$ avec une précision ϵ. On rappelle que

$$\sin(x) = \sum_{n=0}^{+\infty} \left(\frac{(-1)^n x^{2n+1}}{(2n+1)!} \right)$$

Algorithme de $\sin(x)$

Entrées : x, ϵ : réel
Variables : n : entier
uN, x2, sinX : réel
sinX $\leftarrow$ x, n $\leftarrow$ 0
uN $\leftarrow$ x, x2 $\leftarrow$ x * x
tant que $|uN| \geq \epsilon$ **faire**
    uN $\leftarrow$ -uN * x2 / ((2n+2)(2n+3))
    sinX $\leftarrow$ sinX + uN
    n $\leftarrow$ n + 1
écrire sinX

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Syntaxe de Python / Fonction

Calcul de $\sqrt{x}$

Calculer $\sqrt{x}$ à l'aide de la suite : $u_{n+1} = \frac{1}{2}\left(u_n + \frac{x}{u_n}\right)$ et $u_0 = x$

Algorithme de $\sqrt{x}$

Entrées : x, ε : réel
Variables : uN, uN1 : réel
uN $\leftarrow$ x
répéter
    uN1 $\leftarrow$ uN
    uN $\leftarrow$ (uN1 + x/uN1)/2
tant que $|uN - uN1| \leq \varepsilon$
écrire uN

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Syntaxe de Python / Fonction

Entiers particuliers

- Un entier n est divisible par p si le reste de la division de n par p est nul
- Un entier n est premier si il n'est divisible que par 1 et lui même
- Un entier n est parfait si la somme de ses diviseurs est égale à n

Questions

- Écrire une fonction qui dit si un entier est premier
- Écrire une fonction qui dit si un entier est parfait
- Écrire une fonction qui calcule la somme de la somme des diviseurs de tous les nombres inférieurs à N
  Exemple : si $d(n)$ est l'ensemble des diviseurs de n, alors :
  - $d(1) = \{1\}$, $d(2) = \{1, 2\}$, $d(3) = \{1, 3\}$, $d(4) = \{1, 2, 4\}$
  - leur somme $S(4) = 1 + 1 + 2 + 1 + 3 + 1 + 2 + 4 = 15$.
- Écrire un programme qui affiche tous les entiers premiers inférieurs à N puis tous les entiers parfaits inférieurs à N

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Tableaux

Tri de tableaux

On souhaite trier un tableau d'entiers par ordre croissant. Pour cela, on propose d'utiliser une méthode de tri par insertion dont le principe est le suivant :

- On divise le tableau en 2 parties : un partie contenant les éléments déjà triés (à gauche) et une partie contenant les éléments restants à trier (à droite)
- On considère le premier élément non trié du tableau et on l'insère au bon endroit dans la partie triée en décalant les autres éléments du tableau vers la droite

Tableau initial	12	1	5	26	7	14	3	7	2
...	...								
5ème étape (Début)	1	5	12	26	7	14	3	7	2
$7 < 26 \Rightarrow$ Inversion	1	5	12	7	26	14	3	7	2
$7 < 12 \Rightarrow$ Inversion	1	5	7	12	26	14	3	7	2
$7 > 5 \Rightarrow$ Arrêt	1	5	7	12	26	14	3	7	2
5ème étape (Fin)	1	5	7	12	26	14	3	7	2
...	...								

Questions

- Implémenter le tri par insertion.
- Écrire une fonction pour remplir un tableau de N entiers aléatoirement (valeur dans $[-N; N]$), une fonction d'affichage et le programme principal.

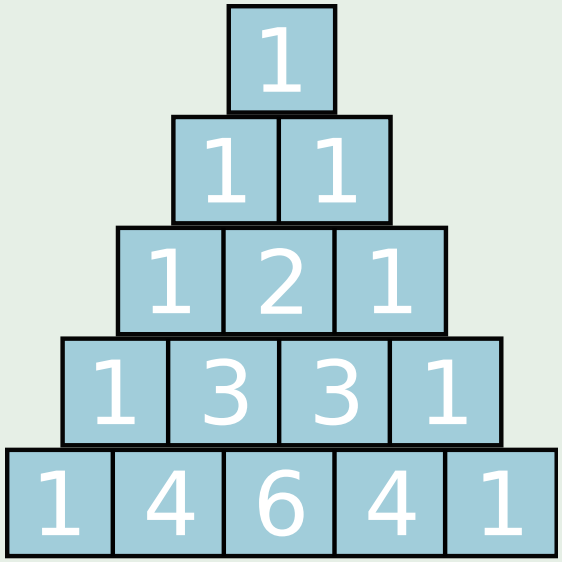
SCIENCES SORBONNE UNIVERSITÉ

Exercices : Tableaux

Triangle de Pascal

- On souhaite calculer et stocker dans une matrice les valeurs des coefficients binomiaux

$$C_n^p = \begin{cases} C_{n-1}^{p-1} + C_{n-1}^{p} & \forall p \in [1; n[\\ 1 \quad \text{si } n = p \text{ ou } p = 0 \end{cases}$$



Question

- Écrire l'algorithme permettant de calculer le triangle de Pascal et l'implémenter
- Écrire le programme principal qui demande à l'utilisateur la valeur de N puis calcule et affiche le triangle de Pascal de rang N.

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Tableaux

1 au début (CodinGame)

- À partir d'une suite de 1 et de 0, on souhaite réunir tous les 1 au début de la liste en un minimum d'opérations.
- Une opération se définit par l'échange de deux éléments situés à des positions différentes.

Exemple : Passer de 0011101010111000000 à 1111111000000000 nécessite 3 opérations

Question

- Écrire un programme qui calcule le nombre minimum d'échanges permettant d'obtenir la liste correctement ordonnée.

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Chaînes de caractères

ASCII Art (adapté de CodinGame, niveau Facile)

- Nous nous intéresserons ici à la représentation des mots en ASCII Art (à l'aide d'un seul caractère #). Par exemple, le mot "Loutre" devient en ASCII Art :

```
#   ### # # ### ##  ###
#   # # # #  #  # # #
#   # # # #  #  ##  ###
#   # # # #  #  # # #
### ### ###  #  # # ###
```

- On dispose d'une fonction `initAsciiTab( H, L )` qui renvoie un tableau contenant la représentation ASCII des 26 lettres de l'alphabet et du caractère '?' (qui représentera tous les autres caractères). Ce tableau a H cases qui contiennent chacune une chaîne de $L \times 27$ caractères.

```
 #  ##   ## ##  ### ###  ## # # ###  ## # # #   # # ###  #  ##
# # # # #   # # #   #   #   # #  #    # # # #   ### # # # # # #
### ##  #   # # ##  ##  # # ###  #    # ##  #   ### # # # # ##
# # # # #   # # #   #   # # # #  #  # # # # #   # # # # # # #
# # ##   ## ##  ### #    ## # # ###  #  # # ### # # # #  #  #
```

SCIENCES SORBONNE UNIVERSITÉ

Exercices : Chaînes de caractères

ASCII Art (adapté de CodinGame, niveau Facile)

- Nous nous intéresserons ici à la représentation des mots en ASCII Art (à l'aide d'un seul caractère #).
- On dispose d'une fonction `initAsciiTab( H, L )` qui renvoie un tableau contenant la représentation ASCII des 26 lettres de l'alphabet et du caractère '?' (qui représentera tous les autres caractères). Ce tableau a H cases qui contiennent chacune une chaîne de $L \times 27$ caractères.

Question

- Écrire un programme demandant une chaîne à l'utilisateur et la convertissant en ASCII Art.