Traduction de la notation algorithmique en langage Python

Ou comment adapter un langage à l'expérimentation d'algorithmes

Jean-Michel Adam – Université Grenoble Alpes – UFR SHS – Département IMSS

Des ressources

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

Sous Linux, Python 2.7 installé par défaut…

Pour installer Python 3.5 :

```
sudo apt-get install idle3
```

Wiki en français:
https://wiki.python.org/moin/FrenchLanguage

Livre gratuit:
Apprendre à programmer avec Python 3
Code source des exemples et les solutions des exercices proposés dans l'ouvrage (Pour Python2 et Python3 - archive ZIP)

Éléments à traduire

- Algorithme principal
- Objets élémentaires définis dans les lexiques
- Agrégats
- Instructions élémentaires
- Analyses par cas
- Itérations
- Actions et Fonctions
- Les tableaux
- La machine-tracés
- Les fichiers séquentiels
- Les classes

Structure générale de l'application

Notation algorithmique

lexique principal

définition des variables de l'algorithme principal et notification des actions et des fonctions utilisées

algorithme principal

texte de l'algorithme

Python

```
# identification auteur, date
# définition de l'application
# lexique partagé
réalisation des actions et fonctions
def principal():
# lexique principal
    lexique de l'algorithme principal
# algorithme principal
    traduction de l'algorithme principal
```

Fichier nomapplication.py

Commentaires

Les commentaires sur une ligne débutent par #

```
# ceci est un commentaire qui finit une ligne

#*********************************************
# Voici un autre commentaire sur             *
# sur plusieurs lignes                       *
# notez l'esthétique particulièrement soignée ;-) *
#*********************************************
```

Types

Un type est caractérisé par :

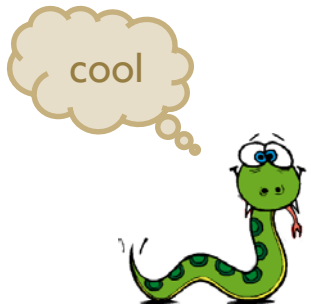
- un ensemble de valeurs
- un ensemble d'opérations définies sur ces valeurs

Python est un langage typé dynamiquement , c'est-à-dire qu'une variable peut changer de type suite à l'affectation d'une valeur d'un autre type.

Type prédéfinis en Pyton :

- entiers : **`int`**
- réels : **`float`**
- booléens : **`bool`**
- chaines: **`str`** un caractère est une chaine de longueur 1
- les tuples : **(** suite d'objets séparées par des , **)**
- les listes : **[** suite d'objets séparées par des , **]**

Le type entier



Valeurs quelconques, pas de limite !

Constantes entières

Les constantes peuvent s'écrire

- en décimal : 12, -23, 2016, …
- en octal précédées de **0o** ou **0O**:
  0o12, - 0O341, 0o777, ...
- en hexadécimal précédées de **0x** ou **0X** :
  0xc, -0X17A, 0xffff …

Les réels en Python

Les réels sont représentés en machine par une valeur approchée en base 2 ; un réel R est représenté par une mantisse et un exposant :

S	exposant	mantisse

- Par défaut, la précision des calculs sur les réels est surprenante en Python:

```
>>> a = 0.1 + 0.2
>>> a
0.30000000000000004
```

Les réels en Python

- Pour obtenir des résultats précis dans les calculs sur les réels il faut utiliser la classe **Decimal** à importer de du module **decimal**

```
>>> from decimal import *
>>> a = Decimal('0.1')
>>> b = Decimal('0.2')
>>> c = a+b
>>> print(c)
0.3
```

Conversion de Types

Conversion explicites en utilisant le nom de type :

```
>>> a = 1234444455039448739822344
>>> a
1234444455039448739822344
>>> b = str(a)
>>> b
'1234444455039448739822344'
>>> c = int(b)
>>> c
1234444455039448739822344
>>> d = c+0.1
>>> d
1.2344444455039449e+26
>>> e = str(d)
>>> e
'1.2344444455039449e+26'
```

Les caractères

- Les caractères sont des **`str`** de longueur 1
- Un caractère est représenté sur 16 bits : codage Unicode de 0 à 65535
- Exemples de constantes caractères : ' ', '0', 'O', "a", '\n' (retour à la ligne), '\t' (tabulation horizontale)
- Les 256 premiers caractères du codage Unicode correspondent à l'ASCII étendu
- Il existe une relation d'ordre sur les caractères qui suit l'ordinal de l'Unicode : tous les opérateurs de comparaison peuvent être appliqués
- Fonctions prédéfinies: **ord** et **chr**
- **ord** renvoie le code d'un caractère donné :

  ord('A') → 65 ord('♉') → 9801
- **chr** renvoie le caractère correspondant à un code :

  chr(65) → 'A' chr(9801) → '♉'

Les caractères

Certains caractères non imprimables ont une représentation particulière :

`'\b'`	retour en arrière
`'\t'`	tabulation
`'\n'`	passage à la ligne
`'\f'`	saut de page
`'\r'`	retour-chariot
`'\"'`	double quotes
`'\''`	apostrophe
`'\\'`	backslash
`'\udddd'`	unicode en hexadécimal
chr(code)	unicode en décimal, hexa ou octal

Les booléens

Les booléens sont de type **`bool`** :

Un booléen ne peut prendre que les valeurs **`True`** et **`False`**

Déclarations de variables

Python étant typé dynamiquement, on peut décrire le lexique par de simples commentaires

Notation algorithmique

```
n : entier      // définition
r : réel        // définition
c : caractère   // défintion
b : booléen     // défintion
ch : chaine     // défintion
```

Python

```
# n : entier      // définition
# r : réel        // définition
# c : caractère   // définition
# b : booléen     // définition
# ch : chaine     // définition
```

Déclarations de variables

On peut aussi décrire le lexique en utilisant le constructeur des classes de base :

Notation algorithmique

```
n : entier      // définition
r : réel        // définition

c : caractère   // défintion
b : booléen     // défintion
ch : chaine     // défintion
```

Python

```
n = int()     # définition
r = float()   # définition
r = Decimal() # définition
c = str()     # définition
b = bool()    # définition
ch = str()    # définition
```

Déclarations de variables

Par défaut, les initialisations de variables sont les suivantes :

```
int()          0
float()        0.0
bool()         False
str()          ''
Decimal()      0
```

Les chaînes

Les chaînes sont représentées en Python par des objets de la classe prédéfinie str.

Une chaîne est une suite de caractères délimitées par des guillemets (doubles ou simples)

```
ch1 = "hello "      # ch1 est initialisée à "hello"
ch2 = 'world'       # ch2 est initialisée à "world"
```

Tous les caractère Unicode sont autorisés

Les constantes nommées sont des variables en Python

Notation algorithmique

PI : le réel 3.14159265

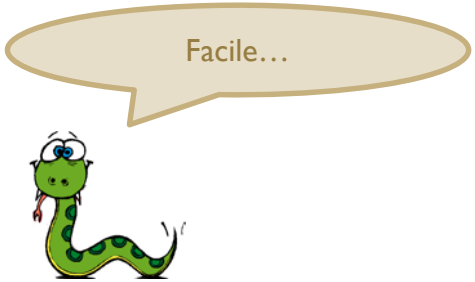
TITRE: la chaîne "DCISS"

OUI: la chaîne "Oui"

MAX: l'entier 9999

Python

```
PI = 3.14159265
TITRE = "DCISS"
OUI = "Oui"
MAX  = 9999
```



Opérations élémentaires

Notation algorithmique

- opérations sur les entiers :

  + - * / div mod

exemple :

a + (b * c) div d + e mod 3

- opérations sur les réels :

  + - * /

  pent(x)

  pdec(x)

Python

- opérateurs sur les entiers :

  + - * / // %

  bonus: ** (puissance)

a + (b * c) // d + e % 3

- opérateurs sur les réels

  + - * /

  int(x)

  x – int(x)

Opérations élémentaires sur les chaînes

Notation algorithmique

- opérations sur les chaînes :

- concaténation : &

  ch & "jour"

- ajout d'un caractère : ∘ •

  'c' ∘ ch

  ch • 'c'

  nième(ch,i)

  longueur(ch)

- sous-chaine :

  souschaine(ch,deb,long)

- deb(ch)
- fin(ch)
- pre(ch)
- der(ch)

Python

- opérateurs sur les chaînes :

- concaténation : +

```
ch + "jour"
```

- ajout d'un caractère : +

```
'c'+ ch
ch +'c'
ch[i]
len(ch)
```

- sous-chaine :

```
ch[deb:deb+long]
ch[:-1]
ch[1:]
ch[0]
ch[-1]
```

Opérations booléennes

Notation algorithmique

opérations sur booléens:

non

et ou

etpuis oualors

comparaisons :

= ≠

< >

≤ ≥

Python

opérations sur booléens:

`not`

`&` `|`

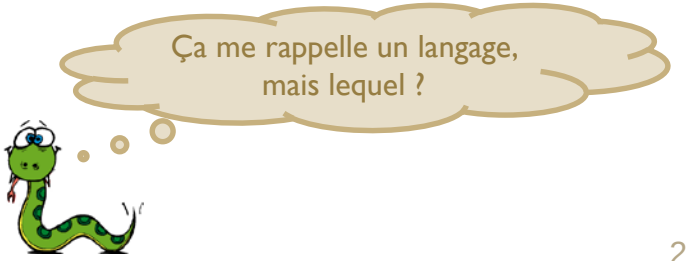
`and` `or`

comparaisons :

`==` `!=`

`<` `>`

`<=` `>=`



Priorité décroissante des opérateurs

Operateur	Description
()	Parenthèses
f(args...)	Appel de fonction
x[index:index]	Sous-liste
x[index]	Sélection d'un élément de liste
x.attribute	Référence d'attribut
**	Puissance
~*x*	Négation bit à bit
+*x*, -*x*	+ et - unaires
*, /, %, //	Multiplication, division, reste (modulo), div
+, -	Addition, soustraction
<<, >>	Décalages de bits
&	ET bit à bit
^	OU exclusif (XOR) bit à bit
|	OU bit à bit
in, not in, is, is not, <, <=, >, >=, <>, !=, ==	Appartenance, identité, comparaisons
not *x*	négation
and	etpuis
or	oualors
lambda	Expression Lambda

Traduction des instructions élémentaires

Notation algorithmique

- affectation :

  v ← expression
- saisie au clavier:

  cl.saisir(x)

  e.afficher("Entrez n: ")

  cl.saisir(n)
- affichage :

  e.afficher("Total : ", t)

Python

- affectation :

  `v = expression`
- saisie au clavier:

```
x = input()      # x est une chaine
x = int(input()) # saisie convertie

n = int(input("Entrez n: "))
```

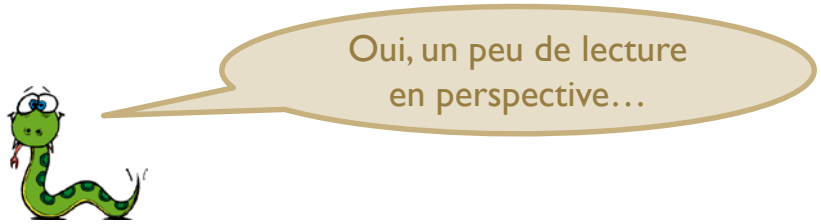
- affichage :

```
print("Total :", t)
```

- affichage sans retour à la ligne :

```
print("Total :", t, end="")
```

Il existe de nombreuses possibilités d'affichage formaté dans Python



Analyse pas cas

Notation algorithmique

selon variables
  condition1 : action1
  condition2 : action2
  condition3 : action3
fselon

Python

```
# selon variables
if condition1:
⇥ action1
elif condition2:
⇥ action2
else: # condition3
⇥ action3
# fselon
```

Les ⇥ sont des caractères de tabulation (4 caractères dans un autre éditeur).

Utiliser un caractère de tabulation par niveau d'emboitement

Exemple

```
selon abréviation
    abréviation  = "M" :     e.afficher("Monsieur ")
    abréviation  = "Mme":  e.afficher("Madame ")
    abréviation  = "Mlle":   e.afficher("Mademoiselle ")
    autrement:                e.afficher("Madame , Monsieur")
fselon
```

```
# selon abreviation
if abreviation == "M":
    print("Monsieur")
elif abreviation == "Mme":
    print("Madame")
elif abreviation == "Mlle":
    print("Mademoiselle")
else: # autrement
    print("Madame, Monsieur")
# fselon
```

Traduction des analyses par cas

Notation algorithmique

si condition
alors actions
fsi

si pre(ch) = 'A'
alors nba ← nba +1
fsi

Python

```
if condition:
   actions
#fsi

if  ch[0] == 'A':
  nba = nba + 1
#fsi
```

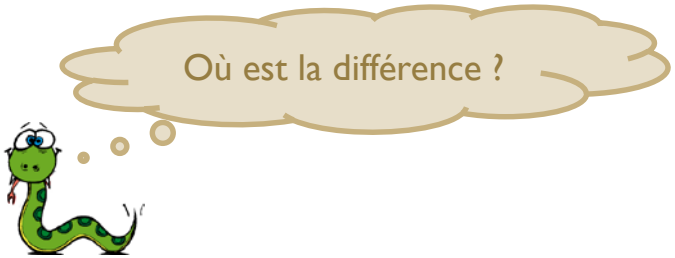
Traduction des analyses par cas

Notation algorithmique

si condition
alors action1
sinon action2
fsi

Python

```
if condition:
    action1
else:
    action2
#fsi
```



Cas particulier d'analyse par cas : le switch

Notation algorithmique

```
selon  v
  v = a :  x ← valeur1
  v = b :  x ← valeur2
  v = c :  x ← valeur3
   ...
  autrement : x ← valeurk
fselon
```

Python

```
switcher = {
    a: valeur1,
    b: valeur2,
    c: valeur3,
    …
}
x = switcher.get(argument, valeurk)
```

Vraiment très particulier, ce cas…

Traduction de l'itération

Notation algorithmique

répéter n fois
    action
frépéter

répéter 4 fois
    s ← s + i
    i ← i + 1
frépéter

Python

```
for _ in range(n):
    action

for _ in range(4):
    s = s + i
    i = i + 1
```

_ est une variable anonyme

range(4) est la même chose que range(0,4) qui correspond à l'intervalle [0,4[

Traduction de l'itération

Notation algorithmique

tantque **ccont** faire
    action
ftq

i $\leftarrow$ 1
tantque i $\leq$ n faire
    e.afficher(i)
    i $\leftarrow$ i + 1
ftq

Python

```
while ccont:
  action
#ftq

i = 1
while i <= n:
  print(i)
  i = i + 1
#ftq
```

Traduction de l'itération

Notation algorithmique

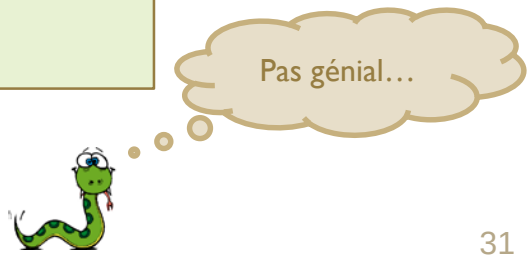
répéter
    actions
jusqu'à carrêt

$i \leftarrow 1$
répéter
    e.afficher(i)
    $i \leftarrow i + 1$
jusqu'à i > N

Python

```
while True:
    actions
    #jusqu'à
    if carrêt: break
#frepeter

i = 1
while True:
    print(i)
    i = i + 1
    if i > N: break
#frepeter
```



Traduction de l'itération

Notation algorithmique

pour i allant de deb à fin pas p
    actions
fpour

Python

```
for i in range(deb, fin+1, p):
    actions
#fpour

deb = 2; fin = 10; p = 2
for i in range(deb, fin+1, p):
    print(i)
#fpour
print("à la sortie i =",i)
```

Par défaut, deb est 0 et p est 1

Traduction des agrégats

Durée : type agrégat
    h : entier ≥ 0 // heures
    m : entier entre 0 et 59 // minutes
    s : entier entre 0 et 59 // secondes
fagrégat

Peut se traduire de différentes manières, le plus simple étant par une classe sans méthodes :

```
class Duree(object):
    # Représentation du type Durée
    def __init__(self, h=0, m=0, s=0):
        # h : entier >=0              // heures
        # m : entier entre 0 et 59  // minutes
        # s : entier entre 0 et 59  // secondes
        self.h = int(h)
        self.m = int(m)
        self.s = int(s)
```

Utilisation des agrégats

Notation algorithmique

d : Durée // définition
d1: Durée // définition
Accès aux champs de d :
d.h ← 12
d.m ← 25
d.s ← 40

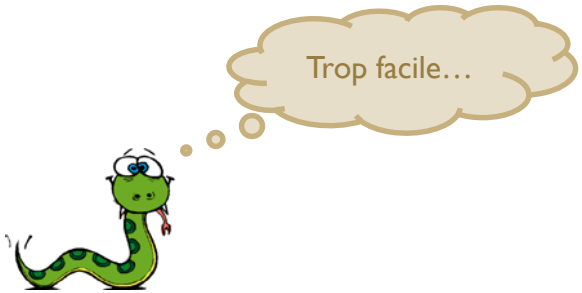
d1 ← Durée(15, 34, 12)

Python

```
d = Duree()  # définition
d1 = Duree() # définition

# Accès aux champs de d :
d.h = 12
d.m = 25
d.s = 40

d1 = Duree(15, 34, 12)
```



Autre exemple

Notation algorithmique

Point: type agrégat

x, y : réels

fagrégat

Utilisation :

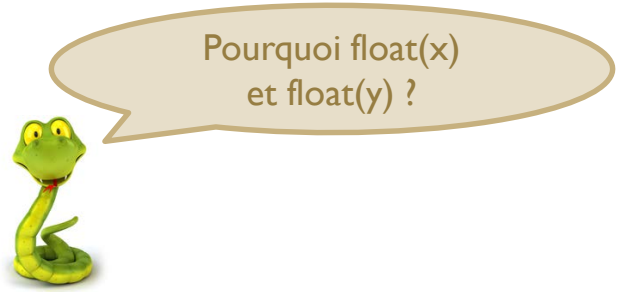
s1 : le Point (1, 12.2)

s2 : Point

Python

```
class Point(object):
# Représentation du type Point
   def __init__(self, x=0.0, y=0.0)
        self.x = float(x) # abscisse
        self.y = float(y) # ordonnée


s1 = Point(1 ,12.2)
s2 = Point()
```



Traduction des tableaux

Notation algorithmique

t : tableau sur [0..NMAX-1] de typeéléments

nbje : tableau sur [0..11] d'entiers =
[31,28,31,30,31,30,31,31,30,31,30,31]

chiffres : tableau sur [0..9] de caractères
= ['0','1', '2', '3', '4', '5', '6', '7', '8', '9']

Python

En Python il n'y a pas de tableau en natif, il n'y a que des listes

Les éléments d'une liste sont indicés à partir de 0

Pour créer un tableau de n éléments, il faut donc construire un liste de n éléments:

```
t = [0] * NMAX
```

Pour les tableaux prédéfinis c'est très simple :

```
nbjm = [31,28,31,30,31,30,31,31,30,31,30,31]
chiffres = ['0','1','2','3','4','5','6','7','8','9']
```

Traduction des tableaux

Notation algorithmique

Désignation d'un élément de tableau :
t[i] désigne l'élément d'indice i du tableau t.

Python

t[i] désigne l'élément d'indice i du tableau t.
Si un tableau est défini sur l'intervalle 1..NMAX dans la notation algorithmique, t[i] dans l'algorithme sera traduit par t[i-1],
à moins d'ignorer l'élément d'indice 0 et de définir t sur l'intervalle 0..NMAX comme ceci :

```
t = [0] * (NMAX+1)
```

Tableaux à plusieurs dimensions

- En Python un tableau est une liste de liste
- Il faut réserver l'espace mémoire représentant ce tableau
- Forme des déclarations de tableaux à plusieurs dimensions :

```
t = [[0] * nbcol for _ in range(nblig)]
```

- Désignation d'un élément de tableau : `t[i][j]`
- Tableau initialisé :

```
CARRE = [[8,1,6],[3,5,7],[4,9,2]]
```

Tableaux à plusieurs dimensions

- Le piège qui tue :


```
>>> lignes, colonnes = 3, 4
>>> lst = [[0] * colonnes] * lignes
>>> lst[1][1] = 2
>>> lst
[[0, 2, 0, 0], [0, 2, 0, 0], [0, 2, 0, 0]]
```

Ce comportement est dû au fait que lorsque Python évalue l'expression **`[[0] * colonnes] * lignes`**, il va interpréter **`[0] * colonnes`** comme étant un objet de type list qui ne sera créé qu'une fois. C'est strictement équivalent à :

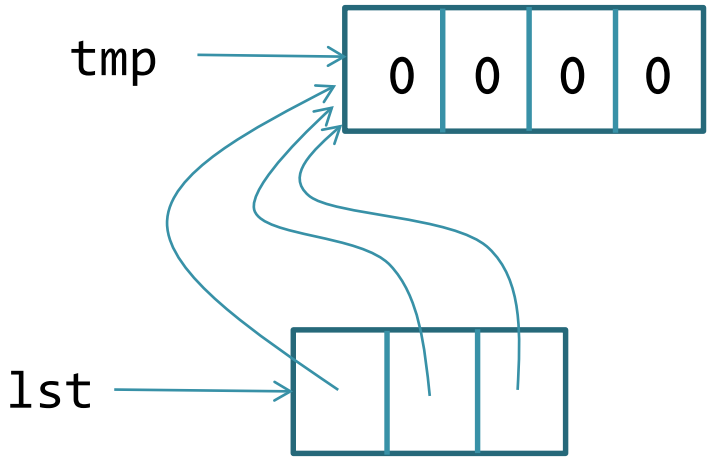
```
>>> tmp = [0] * colonnes
>>> tmp
[0, 0, 0, 0]
>>> lst = [tmp] * lignes
>>> lst
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> lst[1][1] = 2
>>> lst
[[0, 2, 0, 0], [0, 2, 0, 0], [0, 2, 0, 0]]
```

`tmp` est une référence sur une liste, et c'est la référence (et non la liste pointée par `tmp`) qui est répliquée 3 fois dans la nouvelle liste `lst`

Tableaux à plusieurs dimensions

- Le piège qui tue (avec un dessin) :

```
>>> lignes = 3 ; colonnes = 4
>>> tmp = [0] * colonnes
>>> tmp
[0, 0, 0, 0]
>>> lst = [tmp] * lignes
>>> lst
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
>>> lst[1][1] = 2
>>> lst
[[0, 2, 0, 0], [0, 2, 0, 0], [0, 2, 0, 0]]
```



Traduction des fonctions

Notation algorithmique

fonction f(liste paramètres formels) ⟶ type_résultat
// f(X,..) renvoie...
lexique de f
    définition des variables locales
algorithme de f
    texte de l'algorithme
    renvoyer résultat

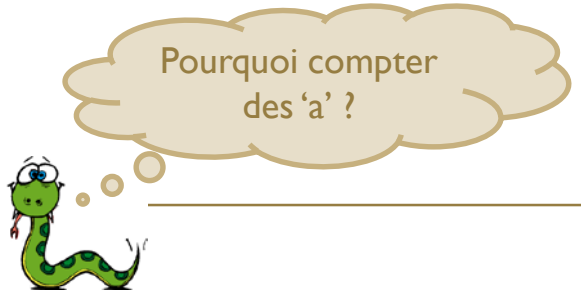
Python

```
def f(liste paramètres formels):
# f(..) renvoie ...
# lexique de f
    définition des variables locales
# algorithme de f
    texte de l'algorithme
    return résultat
```

Exemple de fonction

Notation algorithmique

```
fonction nba(x : chaîne) ⟶ entier ≥ 0
// nba(x) renvoie le nombre de A contenus dans la chaîne x
lexique de nba
    na : entier ≥ 0   // na = nombredea(pp)
    i : entier ≥ 0    // indice de parcours de x
algorithme de F
    i = 0 ; na ← 0
    tantque i < longueur(x) faire
        si nième(x,i) = 'a' ou nième(x,i) = 'A' alors na ← na + 1 fsi
        i ← i + 1
    ftq
    renvoyer (na)
```



Exemple de fonction

Python

```
def nba(x):
 # nba(x) renvoie le nombre de 'a' contenus dans la chaîne x
 # lexique de nba
    na = int()    # Nombre de a de la pp de x
    i = int()     # intermédiaire : indice de parcours de x
 # algorithme de nba
    i = 0
    na = 0
    while i < len(x):
      if (x[i] == 'a') or (x[i] == 'A'):
         na = na + 1
      #fsi
      i = i + 1
    #ftq
    return na
```

Définition des fonctions deb, der, pre, fin, souschaine

Python

```
def pre(x):
# pre(x) renvoie le premier caractère de la chaine x
    return x[0]

def der(x):
# der(x) renvoie le dernier caractère de la chaine x
    return x[-1]

def deb(x):
# deb(x) renvoie le début de la chaine x
    return x[:-1]

def fin(x):
# fin(x) renvoie la fin de la chaine x
    return x[1:]

def souschaine(x,deb,long):
# souschaine de x débutant en deb, de longueur long
    return x[deb:deb+long]
```

Exemples de fonctions

```
def convSD(n):  # --> Durée
# convSD(n) renvoie la durée correspondant à n secondes
 # lexique de convDS
   # n : entier ≥ 0 // paramètre : nbre de secondes à convertir en durée
   d = Duree()  # valeur renvoyée : durée calculée à partir de n
   r = int()    # entier entre 0 et 3599   // intermédiaire : n mod 3600
 # algorithme de convSD
   r = n % 3600
   d.h = n // 3600
   d.m = r // 60
   d.s = r % 60
   return d

def convDS(x): # --> entier >=0
# convDS(x) renvoie le nombre de secondes correspondant à la durée x
 # lexique de convDS
   #   x : Durée          // paramètre : durée à convertir en secondes
   ns = int() # entier > 0 // valeur renvoyée : x convertie en secondes
 # algorithme de convDS
   ns = x.h*3600 + x.m*60 + x.s
   return ns

def sommeD(x, y): # --> Durée
 # lexique de SommeD
   # x,y : Durée   // params : durées dont on veut calculer la somme
   # Fonctions utilisées : convDS, convSD
 # Algorithme de sommeD
    return convSD(convDS(x) + convDS(y))
```

Traduction des actions

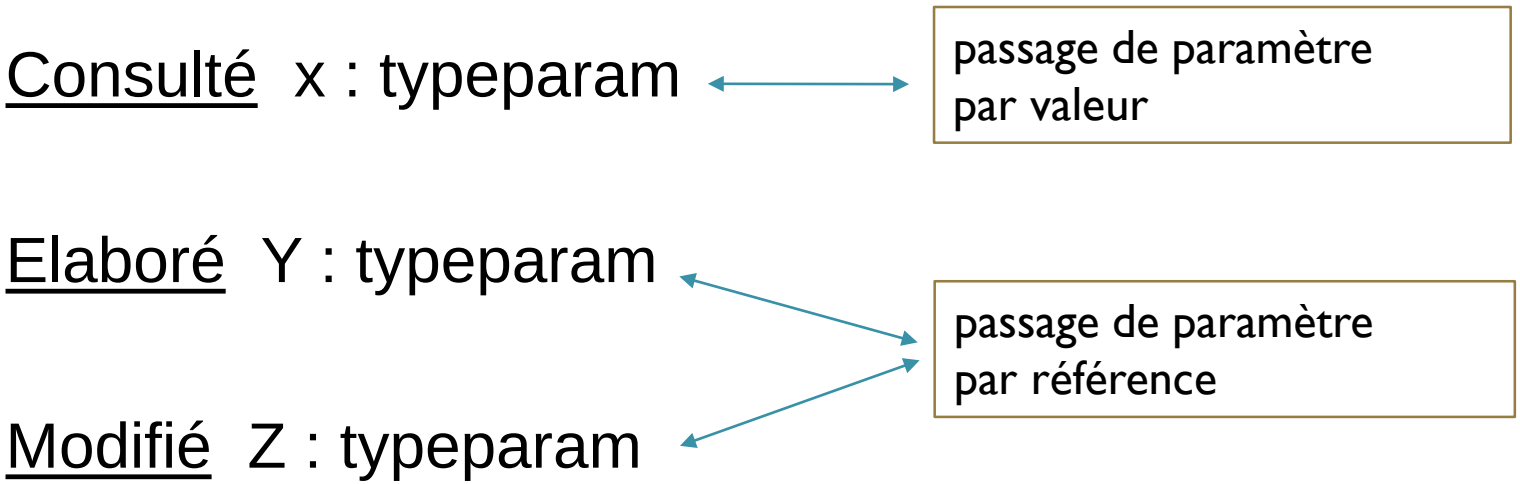
Notation algorithmique

```
action nomAction (liste paramètres formels)
 // Effet :  ...  description de l'effet ...
 // E.I. :    ... description de l'état initial ...
 // E.F. :   ... description de l'état final ...
lexique de NomAction
    définition des variables locales
algorithme de NomAction
    texte de l'algorithme
```

Python

```
def nomAction(liste paramètres formels):
# Effet: ... description de l'effet ...
# E.I.:  ... description de l'état initial ...
# E.F.:  ... description de l'état final ...
# lexique de nomAction
   description des variables locales
# algorithme de nomAction
   texte de l'algorithme
```

Traduction des paramètres



En Python tous les paramètres sont passés par référence, mais certains sont modifiables et d'autres non :

- Les paramètres **non mutables** (`int, float, bool, str,` tuples) ne sont pas modifiables pour l'appelant, ce qui est l'équivalent d'un passage par valeur
- Les paramètres **mutables** (listes, dictionnaires, sets, objets) sont modifiables pour l'appelant, ce qui peut permettre d'implémenter des paramètres élaborés ou modifiés.

Exemples de réalisations de l'action d'échange de 2 valeurs

```
# version 0
a,b = b,a

# fonction échanger
def echanger1(x,y):
    return y,x

# action échanger
def echanger2(par):
    z = par[0]
    par[0] = par[1]
    par[1] = z

# Appel: on transmets une ref
params = [a,b]
echanger2(params)
[a,b] = params
```

```
# autre écriture possible
def echanger3(x,y):
    z = x[0]
    x[0] = y[0]
    y[0] = z

# Appel:
a=[a] ; b=[b]
echanger3(a,b)
[a]=a ; [b]=b
```

Exemples de réalisations de l'action d'échange de 2 valeurs d'objets

```
class Entier:
# objet entier
   def __init__(self,v=0):
        self.v = int(v)

def echanger(x,y):
# échange les valeurs
# de x et de y
    z = x.v
    x.v = y.v
    y.v = z

# Exemple d'appel:
a = Entier(input('a:')))
b = Entier(input('b:')))
echanger(a,b)
```

La machine-tracés

- Une machine-tracés est fournie avec Python : le module turtle
- Elle fonctionne pratiquement comme la machine-tracés étudiée en cours d'algorithmique
- Pour être plus proche de l'écriture des algorithmes vus en cours, on peut se définir une classe qui renomme les différentes actions de manipulation de la machine-tracés.

Equivalences notation algo / Python

Notation algorithmique	Python
`m.vider`	`turtle.reset() ; turtle.up()`
`m.effacer`	`turtle.clear()`
`m.baisser`	`turtle.down()`
`m.lever`	`turtle.up();`
`m.avancer(x)`	`turtle.forward(x)`
`m.reculer(x)`	`turtle.backward(x)`
`m.positionner(s)`	`turtle.setposition(s.x, s.y)`
`m.diriger(a)`	`turtle.setheading(a)`
`m.droite(a)`	`turtle.right(a)`
`m.gauche(a)`	`turtle.left(a)`
`Bonus:`	`Bonus:`
`m.cacher()`	`turtle.hideturtle()`
`m.couleur(c)`	`turtle.color(coul)`
`m.forme(f)`	`turtle.shape(self.shape)`

formes possibles:“arrow”, “turtle”, “circle”, “square”, “triangle”, “classic”
couleurs possibles : 'black','blue','red','green','orange','violet','brown', 'pink', 'gray', etc.

Organisation d'un programme Python

- Une application Python est généralement composée de plusieurs fichiers.
- Chaque fichier correspond à un module pouvant regrouper plusieurs types, classes, fonctions.
- Le programme principal peut être une action nommée du module principal.

Les fichiers séquentiels

Dans la notation algorithmique comme en Python, les fichiers sont des objets disposant d'opérations permettant d'être lus ou créés.

En Python on a deux types de fichiers :

1. Fichiers de texte : éditables avec un éditeur de texte, permettent d'enregistrer et de lire des chaines de caractère
2. Fichiers binaires : illisibles par l'homme, mais pouvant mémoriser tout type d'objet

Les fichiers de texte

Opérations de création d'un fichier de texte :

- Ouverture du fichier en écriture ou en ajout (append)

```
f = open('Nomdufichier.txt','w')
```

ou en ajout (append) :

```
f = open('Nomdufichier.txt','a')
```

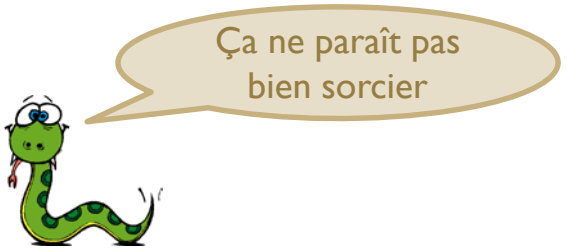
- Enregistrement d'une ligne de texte dans le fichier :

```
f.write("Quel beau temps, aujourd'hui !\n")
```

- Fermeture du fichier

```
f.close()
```

f est un objet fichier



Les fichiers de texte

Opérations de lecture d'un fichier de texte :

- Ouverture du fichier en lecture :

```
f = open('Nomdufichier.txt','r')
```

- Lecture de 1 caractère (chaine de longueur 1), tous les caractères du fichier sont pris en compte ('\n' '\r') :

```
cc = f.read(1)
```

- Lecture de n caractère :

```
ch = f.read(n)
```

- Lecture d'une ligne :

```
ligne = f.readline()
```

- Détection de la fin de fichier : la chaine lue et vide

```
if ch == '':  # on est sur la fin de fichier
```

- Lecture **de toutes les lignes** du fichier :

```
lignes = f.readlines()
```

`lignes` est une liste de chaines

Création et lecture d'un fichier de texte

```
# création d'un fichier de texte
f = open('unfichier.txt','w')
f.write('Bonjour,\n')
f.write("quel beau temps, aujourd'hui !\n")
f.close()
# lecture du fichier créé
f = open('unnfichier.txt','r')
l1 = f.readline()  # lecture de la première ligne
l2 = f.readline()  # lecture de la seconde ligne
l3 = f.readline()  # l3 est la chaine vide
f.close()
```

Fichiers de texte avec des entiers

Les fichiers doivent contenir un entier par ligne

Les fichiers doivent contenir un entier par ligne

```
f = open("entiers.txt","r")
f1 = open("positifs.txt","w")
f2 = open("negatifs.txt","w")
ec = f.readline() # ec est une chaine se terminant par '\n'
while ec !="":
    ec = int(ec)      #conversion ec en un entier
    if ec < 0:
        f2.write(str(ec)+'\n')
    else:
        f1.write(str(ec)+'\n')
    ec = f.readline()
f1.close()
f2.close()
f.close()
print("classement terminé")
```

Fichiers de texte

Autres opérations sur les fichiers de texte :

- Position dans un fichier de texte :

```
p = f.tell()
```

`f.tell()` renvoie la position du fichier f, soit le nombre de caractères de la partie parcourue du fichier.

- Changement de la position du fichier :

```
f.seek(n)   # se place à la position n
```

Les fichiers binaires

Pour utiliser les fichiers binaire il faut utiliser le module pickle :

```
import pickle
```

Opérations de création d'un fichier binaire :

- Ouverture du fichier en écriture ou en ajout (append)

```
f = open('Nomdufichier','wb')
```

ou en ajout (append) :

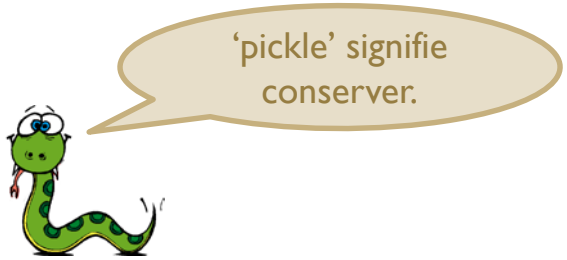
```
f = open('Nomdufichier','ab')
```

- Enregistrement d'un objet dans le fichier :

```
pickle.dump(x, f)   # le type des objets enregistrés est conservé
```

- Fermeture du fichier

```
f.close()
```



Les fichiers binaires

Opérations de lecture d'un fichier binaire :

- Ouverture du fichier en lecture :

```
f = open('Nomdufichier','rb')
```

- Lecture d'un objet :

```
ec = pickle.load(f)
```

- Détection de la fin de fichier :

Lorsque l'on atteint la fin du fichier, l'opération de lecture engendre une exception ! Il faut donc tenir compte de la possible exception en programmant la lecture du fichier :

```
try:
    ec = pickle.load(f)
except EOFError:
    opérations à faire
    si fin de fichier
```

Exemple de création et de lecture d'un fichier binaire

```
# création d'un fichier de 3 objets
a, b, c = 27, 12.96, -12.3
f = open('donnees_test','wb')
pickle.dump(a,f)
pickle.dump(b,f)
pickle.dump(c,f)
f.close()
# relecture et affichage des éléments
# du fichier créé
f = open('donnees_test','rb')
fdf = False
while not fdf:
    try:
        ec = pickle.load(f)
        print(ec, type(ec))
    except EOFError:
        fdf = True
print("fini")
f.close()
```

Classe `Fichier` pour la manipulation de fichiers binaires

Primitives pour la lecture d'un fichier binaire

```
import pickle   # pour les fichiers binaires
class Fichier(object):
    "fichier séquentiel d'objets"
    def __init__(self, nomf):
        # constructeur, nomf = nom du fichier
        self.nomfichier = nomf

    def lirePremier(self):
        self.f = open(self.nomfichier,'rb')
        self.eof = False
        self.lireSuivant()

    def lireSuivant(self):
        try:
            self.eltc = pickle.load(self.f)
        except EOFError:
            self.eof = True

    def ec(self):
        return self.eltc

    def fdf(self):
        return self.eof
```

Classe `Fichier` pour la manipulation de fichiers binaires (suite)

Primitives pour la création d'un fichier binaire

```
    def preparerEnregistrement(self):
        self.f = open(self.nomfichier,'wb')

    def enregistrer(self,x):
        pickle.dump(x, self.f)

    def marquer(self):
        self.f.close()

    def fermer(self):
        self.f.close()
```

Soit `'fichiers.py'` le fichier contenant la classe `Fichier`

Exemple de création et de lecture d'un fichier binaire avec la classe `Fichier`

```
from fichiers import Fichier
# création d'un fichier de 3 objets
a, b, c = 27, 12.96, -12.3
f = Fichier('donnees_test')
f.preparerEnregistrement()
f.enregistrer(a)
f.enregistrer(b)
f.enregistrer(c)
f.marquer()
# relecture et affichage des éléments
# du fichier créé
f.lirePremier()
while not f.fdf():
    print(f.ec(), type(f.ec()))
    f.lireSuivant()
print("fini")
f.fermer()
```

Hiérarchie de classes de fichiers