Introduction à l'algorithmique

Erwan Biland

Stanislas

Année scolaire 2009 - 2010

Objectifs du TD 1

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

↪ Utilisation de la console (*shell*)

- Python calculatrice
- Fichier `machin.py`

↪ Manipulation des variables

- Création/affectation, destruction
- Types `int`, `float`, `str`

↪ Fonctions

- Fonctions `print`, `input`
- Définition d'une fonction, instruction `return`
- Variables globales, variables locales

Débuter avec Python

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

↪ Pour installer Python sur son ordinateur personnel, se rendre sur le site *http ://www.python.org* et télécharger la dernière mise à jour de la version *3.1*. Penser également à télécharger la documentation (tutoriel et autres, au format *pdf*).

Attention : tout ce qui suit concerne *Python 3.1*, et pourra donner des résultats surprenants sous Python 2.6 ou antérieur !

↪ Pour commencer à utiliser Python, lancer l'environnement spécialisé `Idle` qui s'utilise de deux manières :

- Exécuter directement les instructions dans la console ; naviguer dans les instructions précédentes (`Alt+P`, `Alt+N`).
- Écrire les instructions dans un fichier texte (`Ctrl+N`) que l'on sauvegarde (`Ctrl+S`) en choisissant un nom `truc.py` ; enfin, lancer l'interprétation dans la console (`F5`).

Python & Calculatrice

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Python peut être utilisé pour effectuer des calculs élémentaires :

```
>>> 2 + 2
4
>>> (50 - 5 * 6) / 4 # Ceci est un commentaire
5
>>> 3 / 2
1.5
```

On peut également utiliser des décimaux :

```
>>> 4 * 2.5 / 3.3 3.0303030303030303
```

Et même des chaînes de caractères :

```
>>> 'Mais' + 5 * ' euh'
'Mais euh euh euh euh euh'
```

Variables : affectation

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

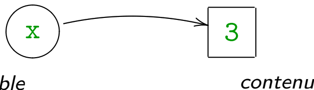
TD 6

Et ensuite ?

Décomposons le traitement par Python de l'instruction suivante, appelée *affectation* :

```
>>> x = 2 + 1
```

1. l'expression 2+1 est *évaluée*, et le résultat obtenu, 3, est stockée dans la mémoire de l'ordinateur ;
2. le nom de variable `x` est inscrit dans la table des noms de Python ;
3. le *nom* `x` est associé à l'emplacement mémoire où est stocké le *contenu* 3.



nom de la variable *contenu*

Le nom d'une variable est formé de lettres, de chiffres, du caractère _ et ne doit pas commencer par un chiffre.

Variables : opérations

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

↪ On peut accéder au contenu d'une variable :

```
>>> 2**x - 2 + x
9
```

↪ On peut connaître l'emplacement où est stocké le contenu d'une variable :

```
>>> id(x)
3586208
```

↪ On peut supprimer une variable :

```
>>> del(x)
>>> x + 2
NameError : name 'x' is not defined
```

Variables : au cours du temps

↪ Un même nom de variable peut être associé successivement à différents contenus :

```
>>> x = 3
>>> x = 2 * x + 1
>>> x = 2 * x + 1
>>> x
15
```

Une seule variable pourra par exemple contenir successivement les termes d'une suite définie par récurrence.

↪ On peut affecter simultanément plusieurs variables.

```
>>> a, b = 3, 'Hello'
>>> a*b
'HelloHelloHello'
```

Fonctions

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Une fonction est un algorithme qui prend des *arguments* en entrée, effectue une séquence d'instructions et affiche ou renvoie un résultat.

```
>>> def <nom_fonction>(<arg_1>,<arg_2,...arg_n>):
...     "Explique cette fonction"
...     <instruction_1>
...     <instruction_2>
...     <instruction_n>
...
>>>
```

- La *définition* d'une fonction commence par `def`, suivi du *nom* de la fonction puis d'une liste entre parenthèses de *paramètres formels*. Cette première ligne se *termine* par `:`.
- Les instructions qui forment le corps de la fonction commencent sur la ligne suivante indentée par une tabulation.

Fonctions : Exemple

Introduction à l'algorithmique
TD 1
Howto
Variables
Fonctions
TD 2
Types nombres
Type chaîne
if
while
TD 3
Complexité
Récursivité
TD 4
Type liste
for, range
TD 5
TD 6
Et ensuite ?

Étant donné un réel x, le programme ci-dessous permet d'afficher la valeur $7 \times x$.

```
>>> def mult_7(x):
...     "Affiche le produit de x par 7"
...     print(7 * x)
...
```

On peut ensuite l'évaluer en tapant

```
>>> mult_7(974)
6818
>>> mult_7(6)+mult_7(2)
42
14
TypeError: unsupported operand type(s) for +:
'NoneType' and 'NoneType'
```

Retourner un résultat

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Pour pouvoir utiliser le résultat d'une fonction comme argument d'une autre fonction ou opérande d'une expression, il faut que la fonction *retourne* une valeur. On utilise `return`.

Pour tout réel x, la fonction ci-dessous permet de retourner $x \times 7$.

```
>>> def mult_7(x):
...     "Retourne le produit de x par 7"
...     return(7 * x)
...
```

On peut ensuite utiliser cette fonction pour de nouvelles aventures. Par exemple,

```
>>> x = mult_7(6) + 10
>>> print(x)
52
```

Variables locales, variables globales

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Pendant l'exécution d'une fonction `fct`, Python crée une table locale des noms, où seront inscrits :

- les noms des *arguments* de la fonction `fct` ;
- les noms des *variables* et *fonctions* créées pendant l'exécution de la fonction `fct`.

Au cours de l'exécution, si Python a besoin du contenu d'une variable `grbl`, il va

1. chercher le nom `grbl` dans la table locale des noms ;
2. si `grbl` ne s'y trouve pas, consulter la table globale ;
3. si `grbl` ne s'y trouve pas non plus, envoyer un message d'erreur.

En général, une fonction ne doit pas faire intervenir de variable globale, sauf à la faire passer en argument.

Règles de bonne programmation

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

1. Réfléchir avant d'écrire.
   - Analyser le problème à résoudre.
   - Déterminer les variables nécessaires à sa résolution.
   - Elaborer un premier schéma d'algorithme.

   Cette première étape se fait sur papier, avant le codage effectif sur l'ordinateur.

2. Ecrire lisiblement
   - Utiliser des noms de variables et de fonctions *explicites*.
   - *Commenter* les différentes étapes du programme grâce au caractère #, documenter les fonctions.
   - Espacer les calculs, utiliser des parenthèses comme vous le feriez à l'écrit.

   Un programme doit pouvoir être lu et relu, par vous-même ou par d'autres.

Objectifs du TD 2

python™

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
`if`
`while`

TD 3
Complexité
Récursivité

TD 4
Type liste
`for`, `range`

TD 5

TD 6

Et ensuite ?

↪ Description des types simples

- Types booléens, nombres, chaînes de caractères
- opérations sur ces types, conversions de types

↪ Instruction conditionnelles

- Instructions *if, elif, else*
- Utilisation de l'indentation pour délimiter des blocs d'instructions

↪ Instructions répétées

- Boucle *while*
- Problème de la terminaison de l'algorithme

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Qu'est-ce-qu'un type ?

Les *objets* susceptibles d'être placés dans la mémoire d'un ordinateur sont de natures différentes. Pour distinguer les uns des autres ces divers contenus, Python fait usage de différents types de variables.

Python a un typage *dynamique* : quand vous assignez une valeur à un nom de variable, Python lui associe automatiquement le type qui correspond au mieux à la valeur fournie.

Pour connaître le type d'une variable, on utilisera la commande `type`.

Types numériques

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Les entiers sont de type `int`.

↪ Les entiers sont définis avec une précision qui n'est limitée que par les capacités de l'ordinateur.

Attention : si un entier commence par `0o` (*resp.* `0x`) il sera lu comme un nombre écrit en base 8 (*resp.* 16).

Types numériques

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Les entiers sont de type `int`.

↪ Les entiers sont définis avec une précision qui n'est limitée que par les capacités de l'ordinateur.

Attention : si un entier commence par `0o` (*resp.* `0x`) il sera lu comme un nombre écrit en base 8 (*resp.* 16).

Le type `float` est utilisé pour représenter les nombres à virgule flottante, c'est-à-dire les nombres décimaux. Une constante de type flottant doit contenir un point `.`, comme le veut la notation anglo-saxonne.

↪ Les nombres flottants sont compris entre 10^{-308} et 10^{308}, avec 12 chiffres significatifs.

Attention : Le nombre `2` sera de type `int` alors que le nombre `2.` sera de type `float`.

TD 1
Howto
Variables
Fonctions
TD 2
Types nombres
Type chaîne
if
while
TD 3
Complexité
Récursivité
TD 4
Type liste
for, range
TD 5
TD 6
Et ensuite ?

Opérations sur les types numériques

Les opérateurs entre nombres sont conformes à l'intuition :

- L'*addition* +, la *soustraction* -.
- La *multiplication* *, la *division* /.
- la *fonction puissance* **, et non ^ !
- La division euclidienne : *quotient* // et *reste* %.

↪ le type du résultat est déterminé par Python en fonction des types des opérandes.

↪ Les opérateurs ont la priorité usuelle, la multiplication ne peut être omise et on utilise les parenthèses pour forcer les priorités.

Plus de maths

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

$\hookrightarrow$ Le type `complex` : $3 + 4i$ est noté `3+4J`, i est noté `1J`.

Plus de maths

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

↪ Le type `complex` : $3 + 4i$ est noté `3+4J`, i est noté `1J`.

↪ Pour utiliser les fonctions mathématiques usuelles, on importe le module `math` de Python via l'instruction, en début de fichier :

```
from math import *
```

commande	sens mathématique
`pi`	π
`e`	nombre de Neper
`exp`	fonction exponentielle
`log`	ln
`log(.,a)`	$\log_a(\cdot)$
`floor`	partie entière

Chaînes : Type

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Les chaînes de caractères sont de type `str`.

Les chaînes peuvent être incluses entre simples quotes (apostrophes) ou doubles quotes (guillemets).

```
>>> "Ceci n'est pas une phrase."
"Ceci n'est pas une phrase"
>>> phrase = '"N\'est-ce pas," répondit-elle.'
>>> print(phrase)
"N'est-ce pas," répondit-elle.
```

La chaîne vide est notée `''` ou `""`.

Pour utiliser des accents dans les chaînes de caractères, il faut inclure, au début de tous vos scripts Python,
`# -*- coding:Latin-1 -*-` ou `# -*- coding:Utf-8 -*-`

Chaînes : Opérations

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Les chaînes peuvent être concaténées avec +, répétées avec *

```
>>> mot = 'Help' + "A"
>>> print('<' + mot*5 + '>')
<HelpAHelpAHelpAHelpAHelpA>
```

La commande `len()` permet de calculer la longueur d'une chaîne de caractères.

```
>>> len('supercalifragilisticexpialidocious')
34
```

Chaînes : Opérations

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Les chaînes peuvent être concaténées avec +, répétées avec *

```
>>> mot = 'Help' + "A"
>>> print('<' + mot*5 + '>')
<HelpAHelpAHelpAHelpAHelpA>
```

La commande `len()` permet de calculer la longueur d'une chaîne de caractères.

```
>>> len('supercalifragilisticexpialidocious')
34
```

On peut accéder directement à un caractère (numérotation à partir de 0) :

```
>>> mot[4]+mot[0]
'AH'
```

On ne peut pas modifier un caractère :

```
>>> mot[0] = 'x'
TypeError : 'str' object does not support item assignment
```

Chaînes : Saucissonnage

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

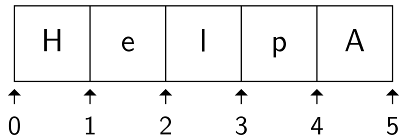
TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?



Les chaînes peuvent être découpées par *saucissonage*.

```
>>> mot[2:4]
'lp'
>>> mot[-4:-2]
'el'
>>> mot[3:1]
''
```

Chaînes : Saucissonnage

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

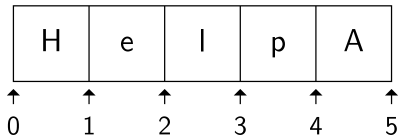
TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?



Les chaînes peuvent être découpées par *saucissonage*.

```
>>> mot[2:4]
'lp'
>>> mot[-4:-2]
'el'
>>> mot[3:1]
''
```

On peut recombiner les morceaux.

```
>>> tom = mot[0:3] + mot[3]
>>> print(tom)
Hell
```

Chaînes : Saucissonnage

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

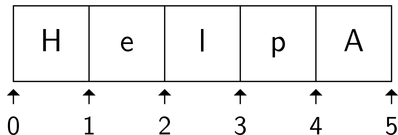
TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?



Un premier index non-défini prend pour valeur par défaut zéro. Un second index omis prend pour valeur par défaut la taille de la chaîne qu'on est en train de découper.

```
>>> print(mot[:2])
He
>>> print(mot[2:])
lpA
```

Chaînes : Saucissonnage

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

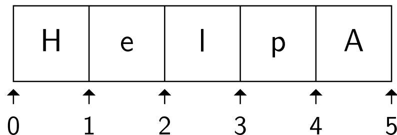
TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?



Un premier index non-défini prend pour valeur par défaut zéro. Un second index omis prend pour valeur par défaut la taille de la chaîne qu'on est en train de découper.

```
>>> print(mot[:2])
He
>>> print(mot[2:])
lpA
```

Un invariant utile des opérations de découpage :

```
>>> motmot = mot[:2] + mot[2:]
>>> motmot == mot
True
```

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions
TD 2
Types nombres
Type chaîne
if
while
TD 3
Complexité
Récursivité
TD 4
Type liste
for, range
TD 5
TD 6
Et ensuite ?

L'instruction if

On construit les boucles conditionnelles à l'aide de l'instruction `if`.

```
>>> if <condition> :
...     <instruction conditionnelle 1>
...     <instruction conditionnelle 2>
...     <instruction conditionnelle 3>
...
>>> <instruction certaine 1>
```

On remarque, comme pour la définition d'une fonction, que c'est l'indentation qui permet de délimiter un bloc d'instructions.
Cette pratique, *obligatoire en Python*, permet au code d'être très lisible. Elle est conseillée dans d'autres langages comme Maple.

if, elif, else

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

On peut enchaîner des instructions conditionnelles :

```
>>> if <condition1> :
...     <instructions1>
... elif <condition2> :
...     <instructions2>
... else :
...     <instructions3>
...
```

Le déroulement de la boucle est le suivant

1. La `<condition1>` est calculée. Si elle est vraie, les `<instructions1>` sont exécutées et la suite ignorée.
2. Si la `<condition1>` est fausse et la `<condition2>` est vraie, seules les `<instructions2>` sont exécutées.
3. Si la `<condition1>` et la `<condition2>` sont fausses, Python exécute les `<instructions3>`.

Le type booléen

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

La condition qui suit une instruction `if` est en fait une *variable booléenne*, qui peut prendre deux valeurs : `True` ou `False`.
Le type d'un booléen (ou variable booléene) est `bool`.

On dispose pour manipuler les booléen des trois *opérateurs logiques* : `or` (ou `|`), `and` (ou `&`), et `not`.

On peut obtenir un booléen à partir d'autres types de variables à l'aide des *opérateurs de comparaison* :

`x == y`	x est égal à y
`x != y`	x est différent de y
`x > y`	x est strictement supérieur à y
`x < y`	x est strictement inférieur à y
`x >= y`	x est supérieur ou égal à y
`x <= y`	x est inférieur ou égal à y
`x is y`	x est la même variable que y

TD 1
Howto
Variables
Fonctions
TD 2
Types nombres
Type chaîne
if
while
TD 3
Complexité
Récursivité
TD 4
Type liste
for, range
TD 5
TD 6
Et ensuite ?

Evaluation des booléens

Les booléens sont évalués intelligemment : le deuxième opérande d'un opérateur n'est évalué que si la valeur du premier est insuffisante pour déterminer le résultat.

```
>>> def test(x):
...     "teste si x est l'inverse d'un entier"
...     return x != 0 and 1/x % 1 == 0
...
>>> print( test(.5) , test(2) , test(0) )
True False False
```

Evaluation des booléens

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Les booléens sont évalués intelligemment : le deuxième opérande d'un opérateur n'est évalué que si la valeur du premier est insuffisante pour déterminer le résultat.

```
>>> def test(x):
...     "teste si x est l'inverse d'un entier"
...     return x != 0 and 1/x % 1 == 0
...
>>> print( test(.5) , test(2) , test(0) )
True False False
```

↪ Les opérateurs non booléens sont prioritaires sur les opérateurs de comparaison, qui sont prioritaires sur les opérateurs logiques.

↪ N'ayez pas peur des parenthèses !

TD 1
Howto
Variables
Fonctions
TD 2
Types nombres
Type chaîne
if
while
TD 3
Complexité
Récursivité
TD 4
Type liste
for, range
TD 5
TD 6
Et ensuite ?

L'instruction while

Si on souhaite exécuter de façon répétitive une séquence d'instructions, on utilise une boucle while.

```
>>> while <condition>:
...     <instruction1>
...     <instruction2>
... <instructions non répétées>
...
```

Le déroulement de la boucle est le suivant

1. on teste la <condition> ;
2. si la <condition> est vraie, on effectue les <instructions> et on recommence la boucle ;
3. si elle est fausse, on sort de la boucle.

Exemple de boucle `while`

La suite de Fibonacci $(f_n)_{n\in\mathbb{N}}$ est définie par :

$$f_0 = 0\,;\ f_1 = 1 \text{ et } \forall n \geq 1,\ f_{n+1} = f_n + f_{n-1}.$$

Écrivons une suite d'instructions qui en calcule les termes inférieurs à 10.

```
>>> a , b = 0 , 1 ; chaine = '0'
>>> while b<10 :
...     chaine = chaine + ' , ' + str(b)
...     a , b = b , a+b
>>> print(chaine)
0 , 1 , 1 , 2 , 3 , 5 , 8
```

Exemple de boucle `while`

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

La suite de Fibonacci $(f_n)_{n\in\mathbb{N}}$ est définie par :

$$f_0 = 0\,;\; f_1 = 1 \text{ et } \forall n \geq 1,\; f_{n+1} = f_n + f_{n-1}.$$

Écrivons une suite d'instructions qui en calcule les termes inférieurs à 10.

```
>>> a , b = 0 , 1 ; chaine = '0'
>>> while b<10 :
...     chaine = chaine + ' , ' + str(b)
...     a , b = b , a+b
>>> print(chaine)
0 , 1 , 1 , 2 , 3 , 5 , 8
```

Attention : Le corps de la boucle doit contenir une instruction qui change la valeur d'une variable intervenant dans la condition évaluée par `while`, pour que cette condition puisse devenir fausse et la boucle se terminer.
↪ Pour forcer l'arrêt, taper `Ctrl + c`.

TD 3

↪ Boucle *while* : approfondissement

- Preuve d'un algorithme : terminaison, justesse du résultat
- Complexité d'un algorithme

↪ Programmation récursive

Complexité

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Afin de comparer plusieurs algorithmes résolvant un même problème, on introduit des mesures de ces algorithmes appelées complexités.

- *Complexité temporelle* : le nombre d'opérations *élémentaires* effectuées par une machine qui exécute l'algorithme.
- *Complexité spatiale* : le nombre de *positions mémoire* utilisées par une machine qui exécute l'algorithme.

Ces deux complexités dépendent de la machine utilisée mais aussi des données traitées.
Pour simplifier, on ne comptera ici que le nombre d'opérations à réaliser (et ce dans le pire des cas).
Dans le cas où le nombre d'opréations dépend d'un paramètre n, il est intéressant de la majorer par une fonction du type : $K \cdot n^{\alpha} \cdot \ln^{\beta} n$.

Complexité : Exemple

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

```
>>> def factorielle(n):
...     p = 1
...     if n <= 1:
...         return(1)
...     else:
...         for k in range(2, n + 1):
...             p = p * k
...         return(p)
...
```

Complexité : Exemple

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

```
>>> def factorielle(n):
...     p = 1
...     if n <= 1:
...         return(1)
...     else:
...         for k in range(2, n + 1):
...             p = p * k
...         return(p)
...
```

Si $n \geq 2$, on effectuera

- 1 affectation en dehors de la boucle.
- 1 comparaison.
- $n - 1$ itérations de la boucle, soit $(n - 1) \times 2$ opérations dans la boucle `for` (multiplication + affectation), 1 affectation initiale de k, $(n - 1) \times 2$ opérations pour l'incrémantation de k (addition + affectation), n comparaisons.
- 1 affichage.

Au total : $5n$ opérations.

Récursivité

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Un programme récursif est un programme qui fait appel à lui-même. On vérifiera toujours qu'une condition est présente pour que le programme termine !
$\hookrightarrow$ Très souvent un algorithme récursif est lié à une relation de *récurrence* permettant de calculer la valeur d'une fonction pour un argument n à l'aide des valeurs de cette fonction pour des arguments inférieurs à n.

Récusivité : Exemple

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

On peut définir x^n par récurrence à partir des relations

$$x_0 = 1 \text{ et } x^n = x * x^{n-1} \text{ si } n > 1.$$

```
>>> def puissance(x,n):
...     if n = 0:
...         return(1)
...     else:
...          return(x*puissance(x, n - 1))
...
```

↪ La machine applique la règle `puissance(x,n)=x*puissance(x,n-1)` tant que $n-1$ est différent de 0, ce qui introduit des calculs intermédiaires jusqu'au cas `puissance(x,0) = 1`. Les calculs en suspens sont alors achevés dans l'ordre inverse pour obtenir le résultat final.

↪ Il faut s'assurer que le cas de base sera atteint en un nombre fini d'étapes.

TD 4

↪ Le type *list*

- Opérations autorisées
- Modification d'un élément, problème de la duplication d'une liste

↪ Parcourir une liste

- Boucle *for*
- Fonction *range*

Le type Liste

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Une liste (type `list`) est déclarée en écrivant ses éléments entre crochets, séparés par des virgules.
La liste vide est notée [].

$\hookrightarrow$ Les éléments ne sont pas nécessairement tous de même type.

```
>>> L = [ 3, 'hello', 2.71 ]
```

Le type Liste

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

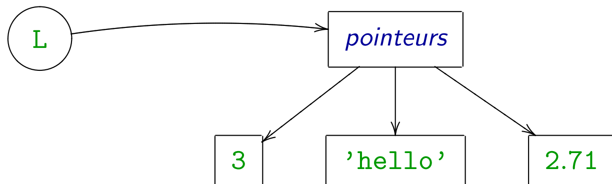
TD 6

Et ensuite ?

Une liste (type `list`) est déclarée en écrivant ses éléments entre crochets, séparés par des virgules.
La liste vide est notée [].

↪ Les éléments ne sont pas nécessairement tous de même type.

```
>>> L = [ 3, 'hello', 2.71 ]
```



Le type Liste

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

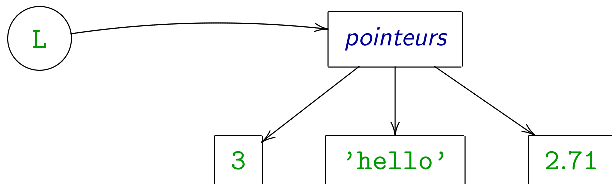
TD 6

Et ensuite ?

Une liste (type `list`) est déclarée en écrivant ses éléments entre crochets, séparés par des virgules.
La liste vide est notée `[]`.

↪ Les éléments ne sont pas nécessairement tous de même type.

```
>>> L = [ 3, 'hello', 2.71 ]
```



Les éléments sont numérotés à partir de 0.

```
>>> L[1]
'hello'
```

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Listes : Opérations

Les listes peuvent se manipuler comme les chaînes de caractères :

- concaténation : `M = L + [ 45, 'bye bye' ]`
- répétition : `MM = M*2`
- saucissonnage : `tranche = MM[2:]`
- longueur : `len(tranche)`

Listes : Opérations

Les listes peuvent se manipuler comme les chaînes de caractères :

- concaténation : `M = L + [ 45, 'bye bye' ]`
- répétition : `MM = M*2`
- saucissonnage : `tranche = MM[2:]`
- longueur : `len(tranche)`

Contrairement aux chaînes, les listes sont modifiables :

- élément par élément : `MM[9] = 'farewell'`
- par tranche : `MM[1:5] = 3 * [ 2, print, M[3:] ]`
- par le bout : `MM.append( 'the end' )`
- et pour faire court : `del( MM[4:-1] )`

Listes : Opérations

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

Les listes peuvent se manipuler comme les chaînes de caractères :

- concaténation : `M = L + [ 45, 'bye bye' ]`
- répétition : `MM = M*2`
- saucissonnage : `tranche = MM[2:]`
- longueur : `len(tranche)`

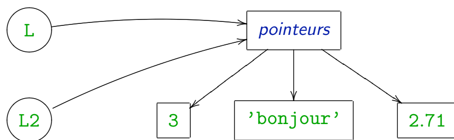
Contrairement aux chaînes, les listes sont modifiables :

- élément par élément : `MM[9] = 'farewell'`
- par tranche : `MM[1:5] = 3 * [ 2, print, M[3:] ]`
- par le bout : `MM.append( 'the end' )`
- et pour faire court : `del( MM[4:-1] )`

Et, oui, une liste ou même une fonction peuvent être des éléments d'une liste...

Affectation et listes

Observons l'effet de l'affectation `L2 = L` :



On peut vérifier ce comportement :

```
>>> L is L2
True
```

On comprend donc que toute modification d'un élément ou d'une tranche de la liste `L2` affectera la liste `L`, et *vice versa*.

```
>>> L[1] = 'bonjour'
>>> L2
[3, 'bonjour', 2.71]
```

Attention : ceci vaut aussi si on passe une liste *en argument* d'une fonction.

Affectation et listes

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

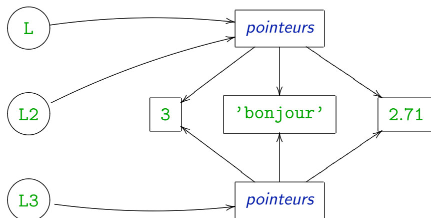
TD 5

TD 6

Et ensuite ?

Si on veut garder une version de L intacte, on préférera en créer une copie :

```
>>> L3 = L[:]
```



On peut alors modifier L3 indépendamment de L :

```
>>> L3[1:2] = [ 'bon', 'jour' ]
>>> L3
[3, 'bon', 'jour', 2.71]
>>> L
[3, 'bonjour', 2.71]
```

Affectation et listes

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

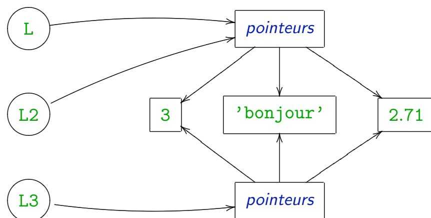
TD 5

TD 6

Et ensuite ?

Si on veut garder une version de L intacte, on préférera en créer une copie :

```
>>> L3 = L[:]
```



Attention : si l'un des éléments de la liste L est lui même une liste (par exemple L[1]), cette méthode permet de *remplacer* L3[1] par un autre objet sans modifier la liste L. Mais elle ne permet pas de modifier un *élément* de la liste L3[1] sans toucher à L[1] !

Exercice : faire un dessin pour comprendre.

For : Définition

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions
TD 2
Types nombres
Type chaîne
if
while
TD 3
Complexité
Récursivité
TD 4
Type liste
for, range
TD 5
TD 6
Et ensuite ?

Lorsqu'on souhaite appliquer successivement une séquence d'instructions à tous les éléments d'une liste, on utilise l'instruction `for`.

```
>>> for <variable> in <liste>:
...     <instructions>
```

Pour calculer la somme de tous les éléments d'une liste `L`, on écrira par exemple :

```
>>> L = [3, 5, 7, 19, 3.14]
>>> somme = 0
>>> for element in L :
...     somme=somme+element
>>> print(somme)
37.14
```

TD 1
Howto
Variables
Fonctions
TD 2
Types nombres
Type chaîne
if
while
TD 3
Complexité
Récursivité
TD 4
Type liste
for, range
TD 5
TD 6
Et ensuite ?

Fonction `range`

La fonction `range(début, fin, pas)` reçoit entre un et trois arguments *entiers*. Elle retourne un objet appelé *itérateur*, qui se comporte (dans une boucle `for`) comme une liste d'entiers, éléments d'une suite arithmétique.

↪ S'il n'y a que deux arguments, le `pas` est fixé à `1` ; s'il n'y a qu'un seul argument, le `début` est de plus fixé à `0`.

```
>>> s = 0 ; chaine = ''
>>> for i in range(5,20,2):
...       s = s + i
...       chaine = chaine + str(i) + ' + '
>>> chaine = chaine[:-2] + '= ' + str(s)
>>> print( chaine )
5 + 7 + 9 + 11 + 13 + 15 + 17 + 19 = 96
```

TD 5

Introduction à l'algorithmique

TD 1
Howto
Variables
Fonctions

TD 2
Types nombres
Type chaîne
if
while

TD 3
Complexité
Récursivité

TD 4
Type liste
for, range

TD 5

TD 6

Et ensuite ?

$\hookrightarrow$ Exercices d'approfondissement

TD 1
Howto
Variables
Fonctions
TD 2
Types nombres
Type chaîne
`if`
`while`
TD 3
Complexité
Récursivité
TD 4
Type liste
`for`, `range`
TD 5
TD 6
Et ensuite ?

TD 6

↪ **Pour les plus avancés :**

- Epreuve de Polytechnique, avec le support de l'ordinateur

↪ **Pour les autres :**

- Exercices d'approfondissement

python™

Perspectives pour la suite de l'année

Introduction à l'algorithmique

TD 1
- Howto
- Variables
- Fonctions

TD 2
- Types nombres
- Type chaîne
- if
- while

TD 3
- Complexité
- Récursivité

TD 4
- Type liste
- for, range

TD 5

TD 6

Et ensuite ?

- Approfondissement Python
- Maple pour les mathématiques
- HTML / CSS
- $\LaTeX$
- ...