Initiation à la programmation avec

Python

langage de programmation

Valérie Bellynck (2009),
d'après un diaporama d'Alain Bozzi (2008),
lui-même venant de celui de Bob Cordeau (2008)

Langage de Programmation Python

Cours/TDs/TPs de 18h

Contrôle

- 1 DS papier

Langage de Programmation Python

But de ce cours

- Connaître et utiliser des schémas algorithmiques simples (parcours, recherches, ...) dans des structures de données différentes (tableaux et fichiers)

- Connaître le langage de programmation Python

- Savoir utiliser les algorithmes de base
  et les adapter à n'importe quel langage

Programme - Script

Un programme ou un script est

- une suite d'instructions s'enchaînant de manière séquentielle pour résoudre un problème donné.

Une instruction est

- Une instruction est composée d'un ou plusieurs mots clés du langage pour effectuer une partie de la tâche globale.

- Une instruction respecte une grammaire et une syntaxe.

Compilation et interprétation

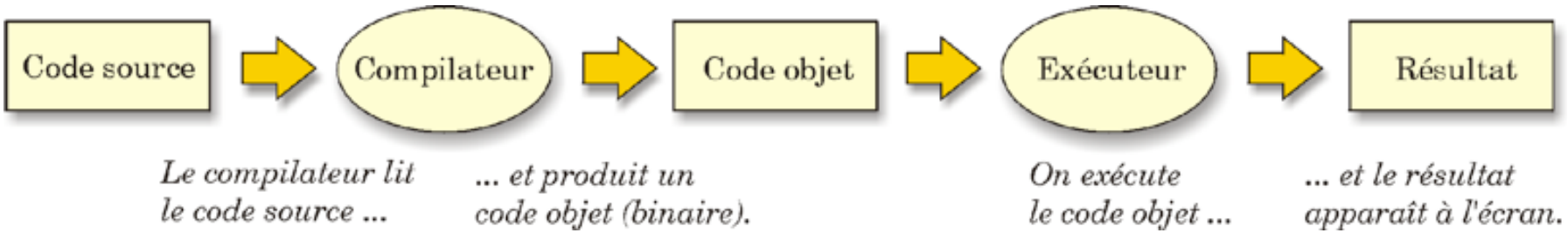
- Le programme est écrit à l'aide d'un logiciel éditeur
  (= sorte de traitement de texte spécialisé)
  est appelé programme source (ou code source).

- Il existe deux techniques principales pour effectuer la traduction d'un programme source en code binaire exécutable par la machine :

  la compilation et l'interprétation

Compilation

- La compilation consiste à traduire la totalité du texte source en une fois.
- Le logiciel compilateur lit toutes les lignes du programme source et produit une nouvelle suite de codes que l'on appelle programme objet (ou **code objet**).
- Celui-ci peut désormais être exécuté indépendamment du compilateur et être conservé tel quel dans un fichier, c'est un fichier exécutable.



Interprétation

- Dans la technique appelée interprétation, le logiciel interpréteur doit être utilisé chaque fois que l'on veut faire fonctionner le programme.
- Dans cette technique en effet, chaque ligne du programme source analysé est traduite au fur et à mesure en quelques instructions du langage machine, qui sont ensuite directement exécutées.

Aucun programme objet n'est généré.

Compilation et interprétation

Chacune de ces deux techniques a ses avantages et ses inconvénients :

- L'interprétation est idéale lorsque l'on est en phase d'apprentissage du langage, ou en cours d'expérimentation sur un projet.
- Avec cette technique, on peut en effet tester immédiatement toute modification apportée au programme source, sans passer par une phase de compilation qui demande toujours du temps.

Par contre, lorsqu'un projet comporte des fonctionnalités complexes qui doivent s'exécuter rapidement, la compilation est préférable.

- Un programme compilé fonctionnera toujours nettement plus vite que son homologue interprété, puisque dans cette technique l'ordinateur n'a plus à (re)traduire chaque instruction en code binaire avant qu'elle puisse être exécutée.

Compilation et interprétation

Certains langages modernes tentent de combiner les deux techniques afin de retirer le meilleur de chacune.

C'est le cas de Python et aussi de Java.

- Lorsque vous lui fournissez un programme source, Python commence par le compiler
- pour produire un code intermédiaire, similaire à un langage machine, que l'on appelle bytecode,
- lequel sera ensuite transmis à un interpréteur pour l'exécution finale.

- Du point de vue de l'ordinateur, le bytecode est très facile à interpréter en langage machine.
- Cette interprétation sera donc beaucoup plus rapide que celle d'un code source.

Les erreurs

1. L’erreur de syntaxe

- Python ne peut exécuter un programme que si sa syntaxe est parfaitement correcte.
- Dans le cas contraire, le processus s'arrête et vous obtenez un message d'erreur.
- Le terme syntaxe se réfère aux règles que les auteurs du langage ont établies pour la structure du programme.

Les erreurs (suite)

2. L'erreur sémantique ou logique

- Le programme s'exécute parfaitement, (pas de message d'erreur) mais le résultat n'est pas celui que vous attendiez.
- Un instruction ou une séquence d'instructions de votre programme ne correspond pas à l'objectif poursuivi. La sémantique (la logique) est incorrecte.
- Rechercher des fautes de logique peut être une tâche ardue.

Exemple : erreur dans la condition du if (***l'interpréteur python la détecte***)
signe d'affectation (erreur) a la place du **signe de comparaison**

```
if(a = 10):
  print "a égal à 10"
else:
  print "a différent de 10"
```

```
if(a == 10):
  print "a égal à 10"
else:
  print "a différent de 10"
```

Les erreurs (fin)

3. L'erreur à l'exécution

Le troisième type d'erreur est l'erreur en cours d'exécution (Run-time error),
qui apparaît seulement lorsque votre programme fonctionne déjà,
mais que des circonstances particulières se présentent
(par exemple, votre programme essaie de lire un fichier qui n'existe plus).

Ces erreurs sont également appelées des exceptions,
parce qu'elles indiquent généralement que
quelque chose d'exceptionnel s'est produit (et qui n'avait pas été prévu).

Vous rencontrerez davantage ce type d'erreur
lorsque vous programmerez des projets de plus en plus volumineux.

Notion d'algorithme

Méthode de conception d'un algorithme

1/ la préparation du traitement :
recherche des **données d'entrées et sorties** nécessaires
à la résolution du problème.

2/ le traitement :
résolution pas à pas du problème posé,
après une décomposition en plusieurs étapes élémentaires
que l'on sait résoudre.

3/ l'édition des résultats :
affichage du traitement afin que
l'utilisateur puisse en prendre connaissance
et traitement des erreurs.

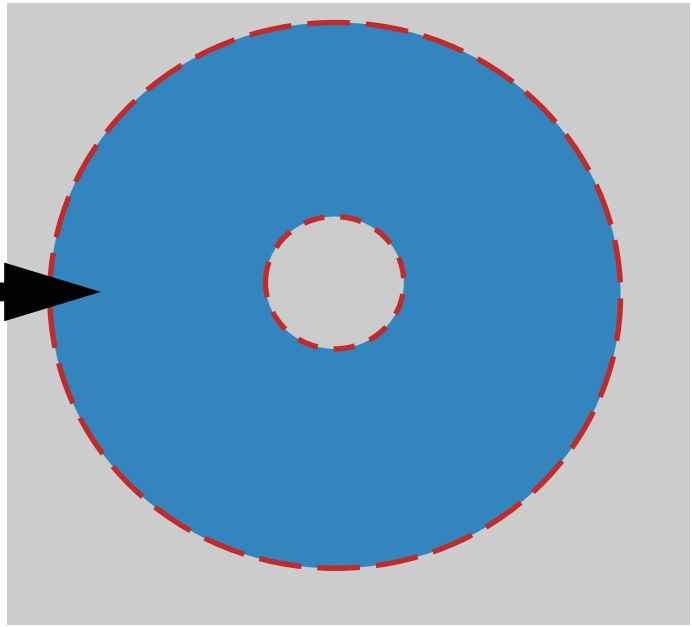
Notion d'algorithme

Etude d'un exemple

Une machine découpe dans une plaque, des disques circulaires de rayon rExt, percés d'un trou circulaire de rayon rInt < rExt et ne débordant pas du disque.

Quelle est la surface d'un disque découpé ?

Surface à calculer



Notion d'algorithme

Démarche analytique

1/ préparation du traitement : quelles sont les données ?

- `pi` est une `*constante` elle ne sera pas demandée à l'utilisateur.
- Les rayons `rExt` et `rInt`, variable d'un disque à l'autre, il faut les demander à l'utilisateur. Ce sont les entrées.

2/ le traitement : calcul de la surface d'un disque

- a) lire les valeurs de `rExt` et `rInt` (lecture au clavier)
- b) calculer la surface du grand disque :

```
sGrandDisque = pi * rExt * rExt
```

`*constante` : variable dont le contenu de change pas pendant toute la durée d'exécution du programme.

Notion d'algorithme

- c) calculer la surface du trou :

```
sDuTrou = pi * rInt * rInt
```

- d) calculer la surface du disque découpé :

```
surface = sGrandDisque – sDuTrou
```

3/ édition du résultat : affichage du résultat

- On va afficher un message, c'est une sortie

```
print "Surface du disque : ", surface
```

La présentation des programmes

Un programme source est destiné à l'être humain.

- Pour en faciliter la lecture,
  il doit être judicieusement commenté.
- La signification de parties non triviales
  doit être expliquée par un commentaire.
- Un commentaire commence par le caractère #
  et s'étend jusqu'à la fin de la ligne :

```
#--------------------
# Voici un commentaire
#--------------------

n = 9 # En voici un autre
```

Les variables

Variable

= conteneur d'information qui porte un nom

= référence à une adresse mémoire (informatiquement)

Les noms des variables sont conventionnellement écrits en minuscule.

Ils commencent par une **lettre** ou le **caractère _**,
puis éventuellement, des lettres, des chiffres.

La **casse est significative**
(les caractères majuscules et minuscules sont distingués).

Ils doivent être **différents des mots réservés** de Python.

Les variables (suite)

Conventions sur les noms en général

- But : donner de la lisibilité sur les noms et limiter les disfonctionnements entre les différentes plates-formes (Unix, Window, Macintosh).

→Jamais de caractères accentués

→Jamais de blanc entre les mots

Syntaxe proposée et appliquée

- Variable : nomDeVariable
- Sous-programme : nomFonction(..)

Exemple

- Variable : valMin
- Sous-programme : ecrireChaine(…)

Les variables (fin)

Typage des variables (spécifique à Python)

- il n'est pas nécessaire de définir le type des variables avant de pouvoir les utiliser.
- il suffit d'assigner une valeur à un nom de variable
  pour que celle-ci soit automatiquement créée
  avec le type qui correspond au mieux à la valeur fournie.
- Par exemple :

n = 10 msg = "Bonjour" euro = 6,55957

Python typera automatiquement ces trois variables :

- n sera de type entier (integer)
- msg sera de type chaine de caractères (string)
- euro sera de type réel (float)

L'instruction type(variable) permet de connaître le type d'une variable

```
print type(n)
```

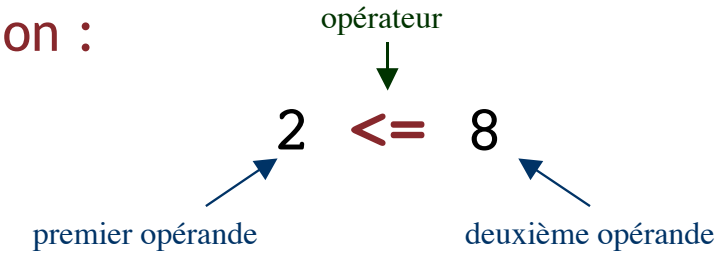
Opérateurs et expression

opérateurs = symboles spéciaux utilisés pour représenter des opérations mathématiques simples, telles l'addition ou la multiplication.

opérandes = valeurs combinées à l'aide des opérateurs.

- Les opérateurs attendent des opérandes de certains types et seulement ceux-là

Exemple d'expression :



- Les opérateurs Python ne sont pas seulement les quatre opérateurs mathématiques de base.
- Il faut ajouter :
  - → l'opérateur ** pour l'exponentiation,
  - → des opérateurs logiques,
  - → des opérateurs agissant sur les chaînes de caractères,
  - → des opérateurs effectuant des tests d'identité ou d'appartenance,
  - → etc.

Les expressions booléennes

Deux valeurs possibles : False, True.

Opérateurs de comparaison : ==, !=, >, >=, <, <=

```
2 > 8                # False
2 <= 8 < 15          # True
```

Opérateurs logiques : not, or, and

```
(3 == 3) or (9 > 24)    # True (dès le premier membre)
(9 > 24) and (3 == 3)   # False (dès le premier membre)
```

Le type entier

Un type caractérise

- la place mémoire nécessaire pour mémoriser les éléments de ce type
- les opérations qu'on peut faire sur les éléments de ce type

Opérations arithmétiques

```
20 + 3    # 23
20 - 3    # 17
20 * 3    # 60
20 ** 3   # 8000
20 / 3    # 6 (division entière)
20 % 3    # 2 (modulo)
```

Les entiers longs (seulement limités par la RAM)

```
2 ** 40      # 1099511627776L
3 * 72L      # 216L
```

Le type entier fait partie des types dits "simples"

Le type flottant

Le type flottant est la seule façon de mémoriser les nombres décimaux et les nombres réels

Les flottants sont notés avec un « point décimal » ou en notation exponentielle :

```
2.718
3e8
6.023e23
```

Ils supportent les mêmes opérations que les entiers, sauf :

```
20.0 / 3          # 6.666666666666667
20.0 // 3 # 6 (division entière forcée)
```

Le type flottant fait partie des types dits “simples”

Les instructions

Instruction = action d'un programme

= opération de base d'un langage de programmation

Les actions courantes dépendent donc du langage et incluent (concernant Python)

- la déclaration de variables et l'attribution de valeurs (définition et affectation),
- le choix d'exécuter une suite d'instructions ou une autre selon une condition,
- l'exécution de boucles itératives ou conditionnelles,
- l'appel de procédures.

L'ordre dans lequel les instructions sont exécutées dans un programme est appelé **flux de contrôle** ou **flux d'exécution**.

Quand on met au point un programme parce qu'il ne fait pas ce qu'on veut,

on simule le flux d'exécution afin de repérer où ce que le programme fait faire est décalé par rapport à ce qu'on aurait voulu.

Le flux de contrôle varie à chaque fois qu'un programme est exécuté, selon les valeurs d'entrée reçues au moment de l'exécution.

L'affectation

On affecte une valeur à une variable en utilisant le signe =

Dans une affectation,
la partie de gauche reçoit ou prend pour valeur la partie droite :

```
a = 2          # prononcez : a "reçoit" 2
               # ou a "prend pour valeur" 2
```

La valeur d'une variable peut évoluer au cours du temps
(la valeur antérieure est perdue) :

```
a = a + 1      # 3 (incrémentation)
a = a - 1      # 2 (décémentation)
```

L'affectation (suite)

Affecter n'est pas comparer !

l'affectation a un effet mais n'a pas de valeur :

```
a = b
# effet : a reçoit la valeur contenue dans b
# valeur de l'expression : aucune
```

la comparaison a une valeur mais n'a pas d'effet :

```
a == b
# valeur de l'expression : True ou False
# effet : aucun
```

Instructions d'affectation : différentes formes

Outre l'affectation simple, on peut aussi utiliser les formes suivantes :

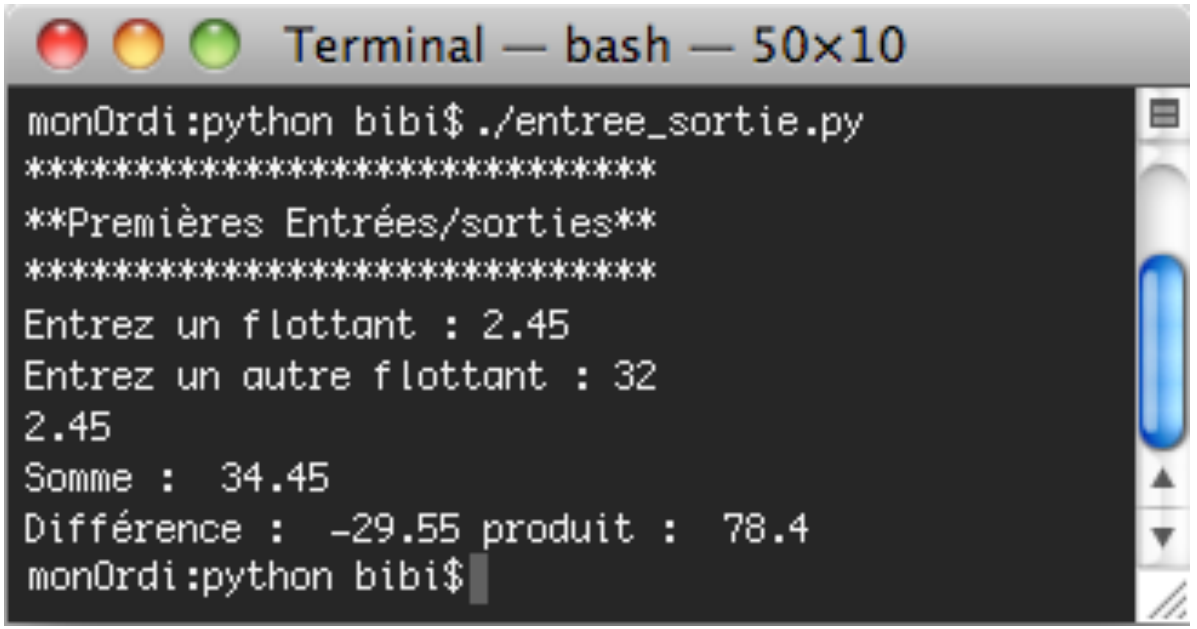
```
a = 4       # forme de base
a += 2      # idem à : a = a + 2 si a existe déjà
c = d = 8           # cibles multiples
                    # (affectation de droite à gauche)
e, f = 2.7, 5.1 # affectation de tuple (par position)
e, f = f, e       # échange les valeurs de e et f
g, h = ['G', 'H'] # affectation de liste (par position)
```

Les entrées / sorties

Il s'agit de **communiquer** avec l'ordinateur
et plus particulièrement avec le programme.

Par exemple,

à l'exécution de votre programme, vous voulez que l'ordinateur
vous demande de saisir une première valeur, puis une deuxième,
et vous affiche la somme des 2 valeurs, leur différence et leur produit



Les entrées

L'instruction **raw_input()** permet d'effectuer une saisie. Le résultat est toujours une chaîne de caractères que l'on peut ensuite transtyper :

```
a1 = raw_input("Entrez un flottant : ")
# a1 contient une chaine (ex :10.52)
a = float(a1) # transtypage en flottant
# ou plus brièvement :
b = float(raw_input(" Entrez un autre flottant : "))
```

Les sorties

L'instruction **print** permet d'afficher des sorties à l'écran :

```
# suite aux entrées précédentes
# par exemple a = 2.45 et b = 32
print a                                  # 2.45
print "Somme : ", a + b                  # 34.45
print "Différence : ", a - b,            # -29.55
print "produit : ", a * b                # 78.4
```

Les mots réservés de python 2.6

(… entourer ceux qu'on a déjà rencontré)

and	def	finaly	in	print	yield
as	del	for	is	raise	
assert	elif	from	lamda	return	
break	else	global	not	try	
class	except	if	or	while	
continue	esec	import	pass	with	

Les modules

Le noyau de base de python contient
des instructions et quelques fonctions de base,
mais par exemple le calcul du sinus risque d'être compliqué…

Un module représente un ou des fichiers python (.py),

- il contient des instructions écrites dans le langage Python.

- But : proposer des services comme la possibilité d'employer :
  → des opérations mathématiques (sin, cos, ...)
  → ou des opérations graphiques (tracer une ligne,...)
  → ou encore bien d'autres opérations spécifiques (base de données, xml, ...)

Les modules (suite)

Dans le langage python pour utiliser un module on écrit au début du programme :

```
from math import *
  # importe la totalité des opérations du module
  # autorise toutes les opérations mathématiques usuelles


from math import sin, pi
  # importe uniquement les opérations sur sin et pi
  # autorise seulement l'utilisation de sin et pi


# Exemple
print sin(pi/4) # 0.70710678118654746


# Pour rappel
print type(pi) # affiche le type de pi => float
```