Cours de

Algorithmique

et langages du Web

Jean-Yves Ramel

Licence 1 – Peip Biologie

Durée totale de l'enseignement = 46h

ramel@univ-tours.fr
Bureau 206 – DI PolytechTours

Organisation de la partie Algorithmique

- 17 ou 18 séances de 2 heures
  - La plupart des séances en salle machine
  - 2 contrôles au moins
- Objectifs
  - Découverte de l'informatique
  - Algorithmique de base
  - Programmation en Python
- Prérequis
  - Aucun

Organisation du cours : partie Algorithmique

- **Au programme….**
  - Ordinateurs et programmatuon
  - Concevoir des programmes de base dans le cadre de la programmation impérative
    - Variables, affectation, expressions
    - Structures de contrôle
    - Chaines de caractères, listes et tableaux
    - Fonctions & paramètres
    - Modules et fichiers
  - Acquérir de bonnes pratiques de programmation
    - lisibilité du code, documentation, méthodologie de test
  - Découvrir le langage Python

Références

Ce support de cours a été construit à partir des sources suivantes :

- Support de cours Licence 1 « Algorithmique et langages du Web » UE 102 - 2011-2012. Auteur : Thomas Devogèle
- Informatique et sciences du numérique. Spécialité ISN en Terminal S. Gilles Dowek. Eyrolles.
- Site OpenClassrooms.com. Tutoriel Python. Sept 2013 http://fr.openclassrooms.com/informatique/python/cours
- Site Débuter avec Python au lycée. Sept 2013. http://python.lycee.free.fr/
- Gérard Swinnen Apprendre Python 3 - http://inforef.be/swi/python.htm
- Cours sur Python 3 - Robert Cordeau
- **…**

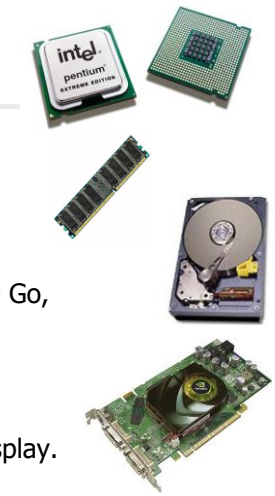
Introduction

Qu'est ce qu'un programme ?

Comment fonctionnent les ordinateurs ?

Qu'est ce qu'un ordinateur ?

Notice d'un ordinateur



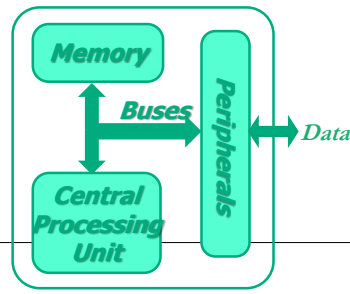
- processeur : Intel® Core™2
  3.33 GHz – bus 1066 MHz
- mémoire : 4 Go
- disque dur : Disque dur DiamondMax 11, 500 Go,
  7200 tpm, buffer 16 Mo
- carte graphique : GeForce 7950 GX2 - 1 Go
- écran : 30-inch Apple Cinema HD Display.

Puissances (en millions d'opérations flottantes par seconde)

- 2880 MFLOPs : Pentium 4 à 3,06 GHz
- 8000 MFLOPs

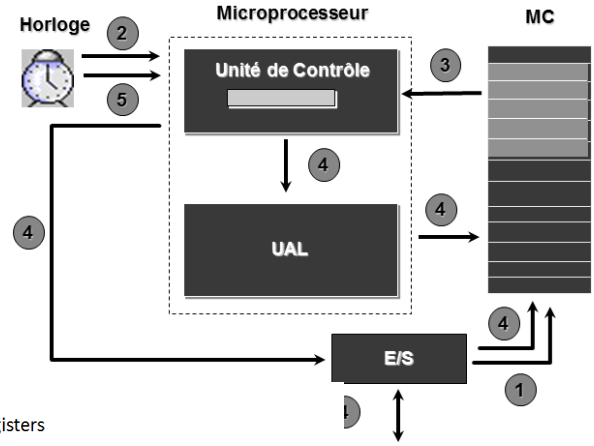
Fonctionnement d'un ordinateur ?

- Un ordinateur stocke en mémoire **des programmes** qu'on lui fournit pour les exécuter
- Un programme = plusieurs petites unités autonomes d'**instructions** qui effectuent des **opérations** sur des **données**
- Un ordinateur possède un répertoire limité d'opérations qu'il sait exécuter très rapidement
- CPU = GPP (General Purpose Processor) ou microprocesseur ou MPU (Micro Processor Unit)

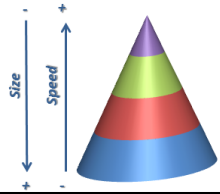


Fonctionnement d'un ordinateur ?

Carte mère ➔



Types de mémoires



- Core Registers
- Cache (L1D/L1P, L2, L3, Cacheable Main Memory)
- Main Memory (SDRAM, DDRAM...)
- Mass Storage (Hard Disk, SD, DVD, Blue-Ray...)

La mémoire vive

- La mémoire est organisée en cellules (ou mots de 32 ou 64 bits)
- Chaque cellule est repérée par son adresse qui permet à l'ordinateur de trouver les informations dont il a besoin
- 2 modes d'accès à la mémoire
  - En lecture : aucun effet sur le contenu
  - En écriture : modifie son contenu
- Caractéristiques
  - Capacité : nombre d'octets
  - Temps d'accès : temps écoulé entre l'instant où l'information est demandée et celui où elle est trouvée (en ms)
- Le contenu de la mémoire (des programmes) est composé
  - de données
  - et d'instructions
    - code de l'opération élémentaire
    - donnée(s) ou adresse des données
- VS mémoire de masse (Disque dur, clé usb, …)

Qu'est ce que l'algorithmique ?

- Résolution informatique d'un problème
  - Le problème (la tâche à résoudre)
  - L'algorithme (la méthode de résolution)
  - Le langage de programmation (langage dans lequel est écrit le programme)
  - L'ordinateur (la machine qui exécute le programme)

L'algorithmique

- Définition 1 : Processus de résolution, par le calcul, d'un problème et permettant de décrire les étapes vers le résultat.
- Définition 2 : Un algorithme est une suite finie et non ambiguë d'opérations permettant de donner la réponse à un problème.
- Exemples :
  - Faire un gâteau au chocolat
  - Changer une roue
  - Résoudre un système d'équations différentielles
  - Trier des informations selon une clé donnée
  - ...

L'algorithmique

- Exercice : Ecrivez la marche à suivre permettant d'accrocher un tableau au centre d'un mur pour cela, vous devez :
  - Définir les objets nécessaires à la résolution du problème
  - Établir la liste des opérations à effectuer
  - Ordonner cette liste

Propriétés d'un algorithme

- Avoir un nombre fini d'étapes
- Avoir un nombre fini d'opérations par étape
- Se terminer après un nombre fini d'opérations
- Fournir au moins un résultat
- Chaque opération doit être :
  - définie rigoureusement et non ambiguë
  - effective, c'est-à-dire réalisable par une machine
- Un algorithme est indépendant du langage de programmation utilisé

Langage de programmation

- Définition
  - Langage informatique permettant à un être humain d'écrire un algorithme (code source) qui sera analysé par un ordinateur
  - Le code source subit ensuite une transformation dans une forme exploitable par la machine, ce qui permet d'obtenir un programme (code exécutable)
- Objectif
  - Faire abstraction des mécanismes de bas niveau de la machine, de sorte que le code source représentant une solution puisse être écrit et compris par un être humain

Langage de programmation

- La programmation
  - Activité de rédaction du code source d'un programme. Elle consiste en la mise en œuvre de techniques d'écriture et de résolution d'algorithmes informatiques
- Un langage de programmation est composé :
  - **d'un vocabulaire** : Ensemble des symboles/termes utilisables pour construire des expressions/opérations
  - **d'une syntaxe** : Ensemble des règles définissant la bonne construction des expressions
  - **d'une sémantique** : Ensemble des règles permettant d'associer un sens à une expression (signification d'une expression)

17

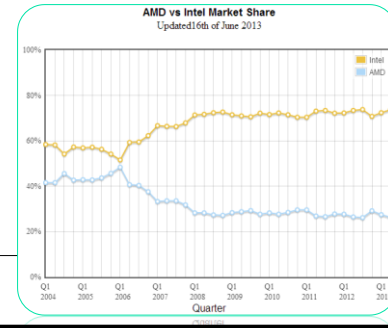
Langage machine

- Un ordinateur ne comprend que le langage machine
- Comment traduire un code source en langage machine ?
- Plusieurs méthodes
  - Interprétation
  - Compilation
  - Méthodes intermédiaires
- Langage Machine
  - Langage natif d'une famille de processeurs
  - Composé d'instructions et de données à traiter **codées en binaire**
  - Chaque processeur possède son propre langage machine donc un code machine ne peut s'exécuter que sur la famille de machine pour laquelle il a été préparé.

18

Langage machine

- Compatibilité
  - Si un processeur A est capable d'exécuter toutes les instructions du processeur B, on dit que A est compatible avec B.
- Exemple de famille
  - La famille x86 regroupe les microprocesseurs compatibles avec le jeu d'instructions de l'Intel 8086 (1978), fonctionne sur tous les processeurs plus récents
- Exemple
  - Core Duo d'Intel
  - Athlon d'AMD
  - Motorola 68000



Langage interprété

- Un programme écrit en langage interprété est converti en instructions directement exécutables par la machine au moment de son exécution.
- Le programme qui traduit le langage interprété est appelé interprèteur; il doit être en fonctionnement sur la machine où l'on veut lancer un programme interprété.
- Quelques exemples de langages interprétés :
  - Python, BASIC
  - PHP, Javascript, Ruby, Perl, etc.
  - Programme = script (langages de scripts)

Langage compilé

- Le programme est traduit une fois pour toute en code machine pour être ensuite exécuté sur la machine cible
- Les code machine obtenu est appelé **exécutable (.exe)**
- **Un compilateur** est un programme informatique qui traduit un programme en langage source vers le langage machine (cible)
  - Le langage source est de haut niveau (destiné à l'humain)
  - Le langage cible est de bas niveau (destiné à la machine)
- Exemple : le compilateur C++ pour windows x86 32 bits
  - Traduit du code C++ en un langage machine destiné à une machine Windows 32 bits

21

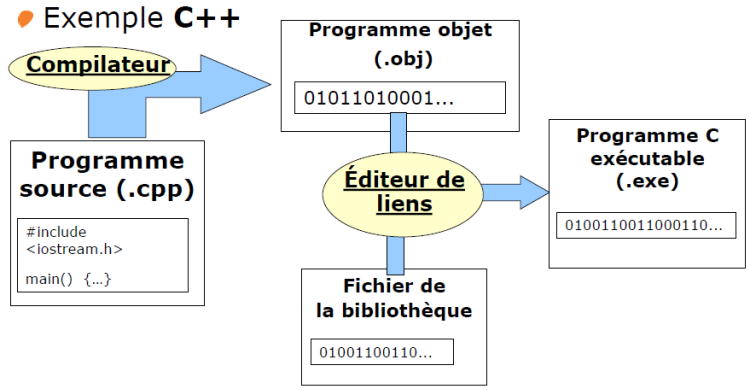
Avantages / Inconvénients

- Langages interprétés
  - Exécution assez lente
  - Code source compatible sur chaque machine ayant un interprèteur
- Langages compilés
  - Exécution assez rapide
  - Code machine spécifique à chaque type de machines et SE
- Méthodes intermédiaires ➔ Machine virtuelle
  - MV = un programme émulant les principales fonctionnalités d'un ordinateur
  - Compilation avec comme langage cible un code exécutable dans une MV. Il faut installer la MV sur la machine cible.
  - **Exemples : Java, Python, C#**
    - Traduction du code Java en bytecode exécutable par la Java Virtual Machine

22

Reprise – Modularité - Evolutivité

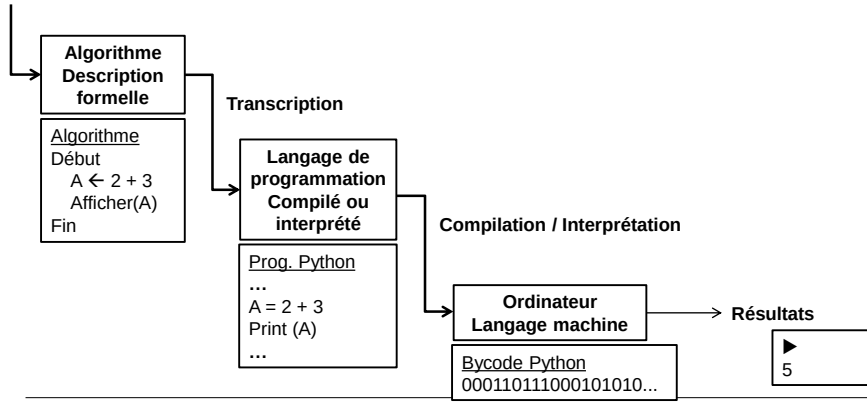
- Il est fondamental de pouvoir ajouter/utiliser dans un nouveau programme, des programmes déjà existants issues de packages ou bibliothèques



Premier plongeon ?

- Problème ➔ Algorithme ➔ Langage Python ➔ Ordinateur

Problème : stocker le résultat de l'expression « 2 + 3 » dans la variable A puis l'afficher



Programmation Procédurale

- Principe
  - Le programme est conçu comme une suite de procédures à réaliser pour répondre au problème considéré
  - Un programme principal fait appel à des fonctions qui peuvent être réutilisées plusieurs fois avec différentes variables
- Exemple
  - Je veux successivement lire deux nombres entrés au clavier
  - Faire un calcul (calcul_R) à l'aide des deux nombres lus
  - Écrire à l'écran le résultat1 du calcul
  - Lire un troisième nombre entré au clavier
  - Faire un calcul (calcul_R) à l'aide du resultat1 et du 3e nombre lu
  - Écrire à l'écran le résultat du calcul 2
- Questions
  - Quelles sont les briques à définir ? Comment les enchaîner ?

25

Programmation Objet

- Principe
  - Un programme crée et manipule des objets
  - Les objets (instances) correspondent à des classes
  - Une classe possède
    - des attributs (variables)
    - des méthodes (procédures traitant les attributs)
- Exemple : On reprend l'exemple précédent
- Questions
  - Quelles sont les briques à définir ? Comment les enchaîner ?
- Réponses
  - Classe Calculateur()
    - Attributs : Valeur1, Valeur2, Resultats
    - Méthodes : Set_Valeur1(…), Set_Valeur2(…), Calcul(), Get_Resultat(), …
  - Création d'instances / d'objets
    - C1 = new Calculateur() ; C2 = new Calculateur() ;

26

En résumé…

- Qu'avez-vous retenu ?
- Donnez un exemple d'algorithme
- Quelle différence entre algorithme et langage de programmation ?
- Quelles differences entre données et instructions d'un programme ?

ANNEXE 1

```
>>> import this
The Zen of Python, by Tim Peters

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

Site web pour ce cours

http://www.rfai.li.univ-tours.fr/WEB_PEIP/index.html

Compte temporaire

Login : epu-guest

Password : Pop2015!

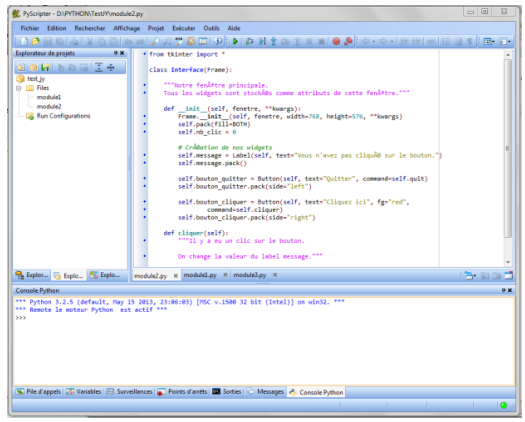
Quand vos comptes PEIP1 seront opérationnels :
Identifiant sous la forme : UNIV-TOURS\2025578t
(votre numéro étudiant suivi de la lettre t)
Mot de Passe : ETU2015 ! (a changer)

Introduction à la programmation avec Python - Partie 1

Environnement, Variables, Opérations,
Contrôle de flux, …

Références Python

- http://www.python.org/
- http://docs.python.org/dev/index.html
- http://portablepython.com/



Portable Python

PyScripter

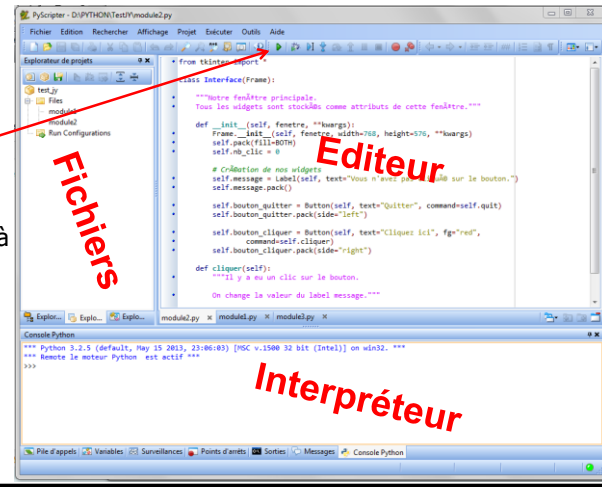
Python – De quoi a-t-on besoin ?

- Installation :
  - une **version de Python**: Attention, les anciennes versions en 2.x sont incompatibles avec les plus récentes en 3.x.
  - Et au minimum, un **éditeur de texte**
  - on écrit un programme dans l éditeur, puis on l'exécute à l'aide d'une commande dans un terminal
- Utilisation d'un **environnement de développement intégré** (IDE) :
  - Sous windows, cela peut être IDLE ou Pyscripter
  - Pack Portable Python : Python + Pyscripter

Pour les TP : http://www.rfai.li.univ-tours.fr/WEB_PEIP/index.html et installer sur votre U:\\.....

Environnement de Dev. – IDE Python

- L'interpréteur : on s'en sert un peu comme une « calculatrice évoluée »
- L'éditeur : on écrit un programme dans l éditeur, puis on l'exécute
- Pleins d'autres outils d'aide à la programmation



Philosophie Python

- Historique
  - Développé en 1989 par Guido van Rossum.
  - En 2005, il a été engagé par Google pour ne travailler que sur Python.
  - Deux principales versions de Python : la 2.7 - la 3.2
- Caractéristiques principales
  - Libre et gratuit
  - Interprété (pas besoin de compilation) ➔ **Script**
  - Interactif (on peut s'en servir comme une calculatrice)
  - Impératif / Fonctionnel / Orienté Objet
  - Extensible (par exemple avec des modules graphiques)
- The Zen of Python
  - Syntaxe claire et cohérente
  - Indentation significative (lisibilité du code)
  - Typage dynamique (pas de déclaration des variables)

Mise au point d'un programme

- **Programmation ➔ Activité complexe ➔ Erreurs**
- **Recherche des erreurs (debogage)**
- Trois types d'erreurs
  - Erreurs de syntaxe ➔ Mauvaise utilisation du langage
  - Erreurs de sémantique ➔ Mauvaise conception (algorithme)
  - Erreur à l'exécution ➔ circonstances particulières (an 2000)
- **Recherche des erreurs et expérimentation**
  - Eviter les erreurs ➔ Réfléchir avant d'agir (conception)
  - Aide à l'écriture (syntaxe) ➔ auto-complétion
  - Correction ➔ enquête policière
  - Activité intellectuelle difficile parfois énervante
- **Des méthodes et outils d'aide existent**

Variables, types, affectations, opérations

- **Une variable** est caractérisée par un nom
  - Attention : les noms sont sans accent, sans espace
  - Majuscule ≠ minuscule
- Renvoie
  - à une **position de la mémoire** dans l'ordinateur ("case")
  - à un **type qui définit la taille** réservée en mémoire et les opérations possibles
- Peut prendre **successivement** différentes valeurs pendant l'exécution d'un programme

```
>>>age = 23
>>>ville = "Tours"
```

Adresse	Case Mémoire	Nom
...	**...**	**...**
10FFA3	**23**	age
10FFA4	**"Tours"**	ville
.....		
10FFA9	**....**	

Variables, types, affectations, opérations

- **Affectations ➔ "="**

```
>>> a = 12345
>>> a = b = 123
>>> a , b = 3 , 5
>>> a , b = b , a
```

Le contenu des variables (cases mémoires) évolue dans le temps (durant l'execution du programme)

- La 1ere affectation s'appelle l'**initialisation** (à ne pas oublier !!!)
- **2 instructions indispensables (liées aux variables)**
  - Afficher le contenu d'une variable
    >>>print(nomvariable)
  - Interroger l'utilisateur pour remplir une variable (affecter une valeur)
    >>> nomvar = input('Entre une valeur : ')

37

Variables, types, affectations, opérations

- **Mots-clés Python** (inutilisables pour nommer les variables)

and	as	assert	break	class	continue	def
del	elif	else	except	False	finally	for
from	global	if	import	in	is	lambda
None	nonlocal	not	or	pass	raise	return
True	try	while	with	yield		

38

Variables, types, affectations, opérations

- **En Python, définition dynamique du Type des variables**
  (interprété VS compilé)

```
>>>a = 12
>>>b = 12.0
>>> type(a)        # Donne le type d'une variable
<class 'int'>
```

- **Les types de base (numériques)**
  - **Nombre Entier (int)**
    >>>a = -12
    codé sur 32 ou 64 bits
  - **Nombre Réel (float)**
    >>>b = 3.14e-10
    codé sur 64 bits
    entre 10^{-308} et 10^{308} avec une précision de 12 chiffres significatifs

39

Variables, types, affectations, opérations

- **Opérateurs possibles sur les variables de types numériques**

Opérations	Symboles	Exemples
addition	+	2 + 5 donne 7
soustraction	-	8 - 2 donne 6
multiplication	*	*6 * 7* donne 42
exponentiation (puissance)	**	*5 ** 3* donne 125
division	/	7 / 2 donne 3.5
reste de division entière	%	7 % 3 donne 1
quotient de division entière	//	7 // 3 donne 2

- **Mémo :**

```
MaxInt = sys.maxsize
–Infini = float('-inf')
```

40

Variables, types, affectations, opérations

- **Les autres types de base**
  - **Les booléens (bool)**

    >>>a = True

    2 valeurs possible : True , False
  - **Les opérateurs sur booléens : and, or, not**

    >>>test = 12 < max **and not** (c == True)
  - **Les opérateurs de comparaison**

```
x == y       # x est égal à y
x != y       # x est différent de y
x > y        # x est plus grand que y
x < y        # x est plus petit que y
x >= y       # x est plus grand que, ou égal à y
x <= y       # x est plus petit que, ou égal à y
```

41

Variables, types, affectations, opérations

- **Les autres types de base** (sur lesquels on reviendra si on a le temps !!!)
  - **Chaine de caractères (str)**

    ch = "Christine" # on peut utiliser " ou `

    Une suite de caractères stockés dans un « tableau »
  - **Opérateurs sur les chaines et caractères spéciaux**

    >>> len(ch) # longueur de la chaine

    >>> ch **+** ch # concaténation

    >>> print(ch[0], ch[3], ch[5]) #accès aux caractères

    C i t # la numérotation commence à 0

>>> Salut = "Ceci est une chaîne plutôt longue\n contenant plusieurs lignes \
... de texte (Ceci fonctionne\n de la même façon en C/C++.\n\
... Notez que les blancs en début\n de ligne sont significatifs.\n«

Variables, types, affectations, opérations

- **Opérateurs sur les chaines et caractères spéciaux**

```
>>>phrase = """Exemple de texte préformaté, c'est-à-dire
…dont les indentations et les
…caractères spéciaux \ ' " sont
…conservés sans
…autre forme de procès."""
```

Problème d'accent ➔ `# -*- coding: utf-8 -*-` / `# -*- coding: latin-1 -*-`

- **Les conversions de types sont possibles (cast)**
  (voire automatique en Python ➔ Danger !)

```
>>>ch = str(1323)
>>>f = float('13.23')  + int('14')
>>>b = a + 12.3     #l'entier a est converti automatiquement en float
```

Variables, types, affectations, opérations

- **Les autres types de base** (sur lesquels on reviendra si on a le temps !!!)
  - **Listes (list) - première approche…**
    Une liste est un ensemble fini d'objets de différents types

```
>>> lst = [1,2,3]
>>> jour = ['lundi', 'mardi', 'mercredi', 180, 20.3, 'jeudi', 'vendredi']
>>> lst = list(range(2,10))      #donne [2, 3, 4, 5, 6, 7, 8, 9]
```

  - **Opérateurs sur les Listes**

```
>>> lst = []                  # creation d'une liste
>>> len(lst)                  # longueur de la liste
>>> lst + lst                 # concaténation
>>> print(lst[0], lst[5])     #accès aux elements
>>> lst.append('voiture')     # ajout en fin de list
>>> del ( lst[4] )            #suppression de l'élément à l'indice 4
>>> lst.pop(4)                # Etc… lst.remove('voiture')
....                          # il y a bien d'autres méthodes !!!
```

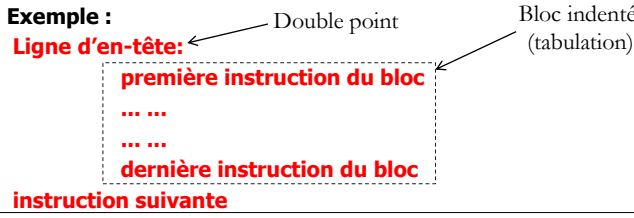
44

Contrôle du flux d'exécution

- **Séquence d'instructions**
  - Les instructions d'un programme s'exécutent **séquentiellement** (les unes après les autres dans l'ordre dans lequel elles ont été écrites)

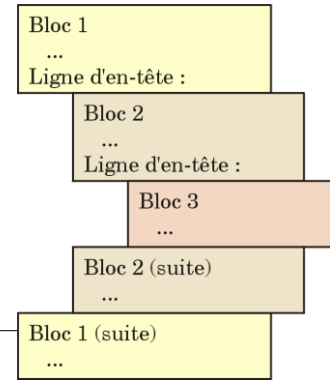
MAIS

- **Instructions composées – blocs d'instructions**
  - Sous Python, les **blocs** d'instructions ont toujours la même structure : une ligne d'en-tête terminée par un **double point**, suivie d'une ou de plusieurs instructions **indentées sous cette ligne d'en-tête**
  - **Exemple :**



Contrôle du flux d'exécution

- **Instructions composées – blocs d'instructions**
  - **La ligne d'en-tête** contient toujours une instruction bien spécifique permettant **de contrôler/modifier le flux d'exécution**
  - **Attention:** toutes les lignes d'un même bloc doivent être indentées exactement de la même manière (c'est-à-dire décalées vers la droite d'un même nombre de tabulations)
  - # Les commentaires sont ignorés



Contrôle du flux d'exécution

- **Instructions conditionnelles (if…elif….else)**
  - Le bloc d'instruction est exécuté sous certaines conditions
  - Exemple :

```
a = 150
if (a > 100) :
    print("a depasse la centaine")
print("a vaut " , a)
```

Attention :

```
a = 150
if (a > 100) :
    print("a depasse la centaine")
    print("a vaut « , a)
```

```
a = 10
if a > 0 :
    print("a est positif")
elif a < 0 :
    print("a est négatif")
else :
    print("a est nul")
print("a vaut « , a)
```


Contrôle du flux d'exécution

- **Instructions conditionnelles imbriquées**

```
if embranchement == "vertébrés":                    # 1
    if classe == "mammifères":                      # 2
        if ordre == "carnivores":                   # 3
            if famille == "félins":                 # 4
                print("c'est peut-être un chat")    # 5
        print("c'est en tous cas un mammifère")     # 6
    elif classe == "oiseaux":                       # 7
        print("c'est peut-être un canari")          # 8
print("la classification des animaux est complexe") # 9
```

- **Rappel opérateurs de comparaison (cf Booléens)**

```
x == y       # x est égal à y
x != y       # x est différent de y
x > y        # x est plus grand que y
x < y        # x est plus petit que y
x >= y       # x est plus grand que, ou égal à y
x <= y       # x est plus petit que, ou égal à y
```

Contrôle du flux d'exécution (if)

- **Exercice 1 :** Ecrire un script qui calcule l'indice de masse corporelle (IMC) d'un adulte et qui en donne l'interprétation (corpulence normale, surpoids...). Cf wikipedia

- **Exercices 2 :** Ecrire un script qui résout l'équation du second degré

```
>>>
Résolution de l'équation du second degré : ax² + bx + c = 0
Coefficient a ? 1
Coefficient b ? -0.9
Coefficient c ? 0.056
Discriminant :  0.586
Deux solutions :
0.0672468158199
0.83275318418
```

```
Résolution de l'équation du second degré : ax² + bx + c = 0
Coefficient a ? 2
Coefficient b ? 1.5
Coefficient c ? 4
Discriminant :  -29.75
Il n'y a pas de solution.
```


Contrôle du flux d'exécution

- **Instructions répétitives**
  - **Répétitions en boucle – L'instruction while**
  - Une **boucle while** permet de **répéter** un certain nombre de fois (voire indéfiniment) une série d'instruction

```
While (condition) :                  #Tant_que la condition est vraie
    bloc d'instructions              # attention à l'indentation
#suite du programme
```

  - **Exemple :**

```
a = 0
while (a < 10):                  # (n'oubliez pas le double point !)
    a = a + 1                    # (n'oubliez pas l'indentation !)
    print(" a vaut : " , a)
#fin de la boucle while
```

Contrôle du flux d'exécution

- **Répétitions en boucle – L'instruction while**
  - La variable évaluée dans la condition doit exister au préalable (il faut qu'on lui ait déjà affecté au moins une valeur)
  - Si la condition est fausse au départ, le corps de la boucle n'est jamais exécuté.
  - Si la condition reste toujours vraie, alors le corps de la boucle est répété indéfiniment
  - Il faut donc veiller à ce que le corps de la boucle contienne au moins une instruction qui change la valeur d'une variable intervenant dans la condition évaluée par while, de manière à ce que cette condition puisse devenir fausse et la boucle se terminer.
  - **Exemple de boucle sans fin (à éviter !) :**

```
n = 3
while n < 5:
    print("hello !")
```


Contrôle du flux d'exécution (while)

- **Exercice 1 :** Ecrire un script qui calcule la moyenne d'une série de n notes. On pourra utiliser une variable qui contient la somme intermédiaire des notes.

- **Exercices 2 :** Ecrire le script du jeu de devinette suivant :

Remarque : pour créer un nombre entier aléatoire entre 1 et 100 :

```
import random
nb = random.randint(1,100)
```

Prévoir la question « voulez vous rejouer ? »

```
Le jeu consiste à deviner un nombre entre 1 et 100 :

--->   50
trop petit !
--->   75
trop petit !
--->   87
trop grand !
--->   81
trop petit !
--->   84
trop petit !
--->   85
Gagné en 6 coups !
```

Contrôle du flux d'exécution

- **Instructions répétitives**
  - **Répétitions en boucle – L'instruction For**
  - Une **boucle For** permet de **répéter** un nombre de fois connu à l'avance une série d'instruction
  - Les éléments de la séquence sont issus d'une chaîne de caractères ou bien d'une liste

```
for (élément) in (séquence) :
        bloc d'instructions
# suite du programme
```

  - **Exemple :**

```
liste = ['Pierre',67.5,18]
for elmt in liste :
        print elmt              # elmt est la variable d'itération
print 'Fin de la boucle'
```


Contrôle du flux d'exécution

- **Répétitions en boucle – L'instruction For**
  - Si le nombre d'itérations à effectuer est connu, il faut utiliser de préférence une boucle **for.**
  - On utilise **while** dans les autres cas
  - For est souvent associé avec la fonction range() pour créer des séquences automatiques de nombres entiers

```
for i in range(1,5):            # Attention range(1,5) renvoie [1,2,3,4]
      print (i)
print ('Fin de la boucle')
```

Contrôle du flux d'exécution

- **Instructions répétitives - break et continue**
  - L'instruction **break** permet de sortir directement d'un bloc d'instruction sans condition (sortie d'une boucle)
  - L'instruction **continue** permet de passer à l'itération suivante dans une boucle (sans passer par les instructions qui suivent dans le bloc en cours)
  - **Exemple :**

```
import time     # importation du module time
while True:
    # strftime() est une fonction du module time
    print ('Heure courante ', time.strftime('%H:%M:%S') )
    quitter = input('Voulez-vous quitter le programme (o/n) ? ')
    if quitter == 'o':
        break
print 'A bientôt'
```


Contrôle du flux d'exécution

- **Pas toujours facile à suivre…**

```
#Suite de Fibonacci
a, b, c = 1, 1, 1
while c < 11 :
    print(b, end =" ")
    a, b, c = b, a+b, c+1
```

- Une représentation des variables est parfois fort utile.
- CF debuggueur

Variables	a	b	c
Valeurs initiales	1	1	1
Valeurs prises successivement, au cours des itérations	1 2 3 5 …	2 3 5 8 …	2 3 4 5 …
Expression de remplacement	b	a+b	c+1

Bilan 1e partie : contrôle

- **Bilan et remarques (1ere partie)**
  - Les variables et leurs types (int, float, string, …)
  - Les fonctions PRINT et INPUT
  - Instructions conditionnelles (IF)
  - Les boucles (FOR, WHILE, Compteur, …)
- **Démarche vue en cours**
  - Identification des variables
  - Mise en place de l'algorithme
  - Codage python
- **Prêt pour le contrôle ? Questions ?**

Introduction à la programmation avec Python - Partie 2

Fonctions, Modules, Fichiers,
Organisation d'un programme, …

Procédures et Fonctions

- Lors de la conception d'algorithmes, le problème posé (complexe) doit souvent être décomposé en sous-problèmes plus simples
- On conçoit alors les algorithmes résolvant les sous-problèmes plus simples stockés et conservés dans des **procédures et fonctions**
- il s'agit d'une même séquence d'instructions qui doit être utilisée à plusieurs reprises éventuellement avec/sur des données différentes **(paramètres)**
- Cela revient à ajouter de nouvelles instructions au langage de programmation (cf print())

Définition d'une nouvelle procedure / fonction (en Python)

```
def nomDeLaFonction (liste de paramètres) :
        """Doc-String = commentaire qui s'affichera lors de la commande help """
        ...
        bloc d'instructions
        ...
        return Resultat
```

Facultatif

- **Appel (Utilisation)**

```
nomDeLaFoction(données)
```

Procédures et Fonctions

Définition d'une Procédure / Fonction simple sans paramètre

```
def table7() :
        n = 1
        while n <11 :
            print(n * 7, end =' ')
            n = n +1
```

Utilisation, il suffit de l'**appeler** par son nom dans le programme :

```
table7()
```

Remarque 1: Dans un programme, la définition des fonctions doit précéder leur utilisation

---

Remarque 2 : La variable spéciale __name__ (et __main__) permet de déterminer si un fichier contient un programme principal ou bien uniquement des définitions de fonctions (fichier module)

```
# test importation ou exécution du script
if __name__ == "__main__":
    # Code du corps du programme (main) à exécuter lorsqu'il ne s'agit pas d'un import
```

Les paramètres

Procédure et Fonction avec paramètres

- Un paramètre peut être vu comme une variable particulière. On lui choisit un nom que l'on place entre les parenthèses qui accompagnent la définition de la fonction.

Exemple

```
def table(base):
   n = 1
   while n <11 :
        print(n * base, end =' ')
         n = n +1
```

Utilisation = Appel

```
table(7)
```

Les paramètres

Fonction avec plusieurs paramètres

```
def tableMulti(base, debut, fin):
    print('Fragment de la table de multiplication par', base, ':')
    n = debut
    while n <= fin :
        print(n, 'x', base, '=', n * base)
        n = n +1
```

Utilisation

```
tableMulti(8, 5, 15)
```

Exercice 1:

Définissez une procédure **ligneCar(n, ca)** qui affiche **n** fois le caractères **ca**.

Ecrivez un programme permettant à l'utilisateur d'exploiter cette fonction en combinaison avec la fonction input()

Procédures et Fonctions

Vraie fonction et procédure

- Une vraie fonction doit renvoyer un résultat via l'instruction **return**
- Sinon on parle plutôt de **procédure** et non pas de fonction

```
def multiplication(n1,n2):
    result = n1*n2
    return result          # la fonction renvoie un résultat

#Prgm principal
valeur = multiplication(120,16)
vf = valeur + 100
print (vf)
```

Exercice 2:

Ecrire un programme exploitant une fonction Str2List(…) transformant une chaine de caractères passée en paramètre une en liste de caractères

Procédures et Fonctions

Exercice 3 : Ecrire une fonction Table(base) qui renvoie une liste contenant les 10 premiers multiples de base

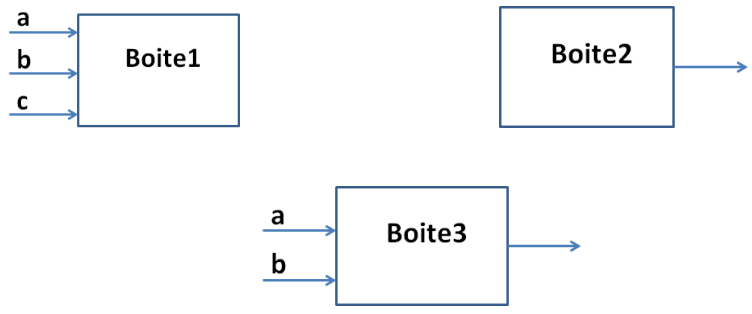
Exercices

Exercice 4:

Ecrire un programme exploitant 2 fonctions que vous définirez vous même, nommées MonMin et MonMax recherchant le minimum et le maximum dans une liste (passée en paramètre)

Représentations graphiques

- Flèches = paramètres et valeur renvoyée par le bloc



Variables locales – Variables globales

Variables locales, variables globales

- Les variables définies à l'intérieur du corps d'une fonction ne sont accessibles qu'à l'intérieur de la fonction elle-même. On parle de **variables locales** à la fonction
- Chaque fois que la fonction est appelée, Python réserve pour elles (dans la mémoire) un nouvel *espace (de noms)* dans lequel le contenu des variables est stocke.
- Cet espace réservé est détruit lorsque l'on sort de la fonction.
- Les variables définies a l'extérieur d'une fonction sont des **variables globales.**
- Leur contenu est **visible** de l'intérieur d'une fonction, mais on ne peut **pas le modifier**
- Pour accéder à (modifier) une variable globale à l'intérieur d'une fonction il faut utiliser le mot clé **global**

Exemple (de non modification)

```
def mask():
    p = 20
    print(p, q)

p, q = 15, 38
mask()
print(p, q)
```

```
20 38
15 38
```

Exemple (d'utilisation du mot-clé global)

```
def monter():
global a      # Python ira chercher la variable globale a
  a = a+1     # au lieu de créer une nouvelle variable
    print(a)

a = 15
monter()
monter()
```

```
16
17
```

Variable locales – Variable globales

```
def toto():
    a=1       # Variable locale
    print(a)

#debut prg principal
a=2           # Variable globale
toto()
print(a)
```

```
1
2
```

```
def toto():
    global a  #facultatif
    print(a)  #car utilisation

#debut prg
a=2
toto()
print(a)
```

```
2
2
```

```
def toto():
    global a  #non facultatif
    a = 1     #car modification
    print(a)

#debut prg
a=2
toto()
print(a)
```

```
1
1
```

68

Variables locales – Variables globales

- **Exercice :** Ecrire 2 fonctions calculant et retournant la valeur de n! et illustrant la différence entre paramètre et variable globale. Ecrire le programme principal exploitant ces fonctions.
- Commentez, expliquez
  - Factoriel(n)
  - Factoriel()

On passe au TD3 – exercice 1

Fonction testannee bissextile

On passe au TD3 – exercice 1

Procedure sans paramètre

Encore plus loin avec les parametres

Typage dynamique des paramètres

- Le mécanisme de typage dynamique fonctionne aussi pour les paramètres d'une fonction

```
def afficher3fois(arg):
    print(arg, arg, arg)
>>> afficher3fois(5)
>>> afficher3fois("Hello")
```

Valeurs par défaut des paramètres

```
def politesse(nom , titre ='Monsieur'):
   print("Veuillez agréer ,", titre , nom, ", mes salutations cordiales.")
>>> politesse('Dupont')
>>> politesse('Durant' , 'Mademoiselle ')
```

Remarque : les paramètres sans valeur par défaut doivent précéder les autres dans la liste.

Paramètres avec étiquette (modification de l'ordre des paramètres)

```
def oiseau(voltage=100, etat='allumé', action='danser la java'):
    print('Ce perroquet ne pourra pas', action)
    print('si vous le branchez sur', voltage, 'volts !')
    print("L'auteur de ceci est complètement", etat)
>>> oiseau(etat='givré', voltage=250, action='vous approuver')
```

Modules et importation de fonctions

Les modules sont des fichiers qui regroupent des ensembles de fonctions

- Les fonctions intégrées au langage sont relativement peu nombreuses : ce sont seulement celles qui sont susceptibles d'etre utilisées très fréquemment.
- Les autres doivent être regroupées dans des fichiers séparés➔ les *modules*
- Pour pouvoir utiliser ces fonctions, il vous suffit d'importer les modules nécessaires :

Exemples

```
import math (as mathematique)
from math import *
from math import sqrt
>>> a = math.sqrt(25)
```

Modules préprogrammés

- Il existe un grand nombre de modules préprogrammés qui sont fournis avec Python.
- Le module *math*, par exemple, contient les **définitions** de nombreuses fonctions mathématiques telles que *sinus*, *cosinus*, *tangente*, *racine carrée*, etc.

Modules personnels

- il est commode de découper un programme important en plusieurs fichiers de taille modeste pour en faciliter la maintenance.

73

TD3 – Où en êtes vous ?

- On reprend et on continue…
- On corrige Médiathèque (version basique)

74

TD3

```
#Definition des fonctions
def saisie_item() :
        ...
def afficherM(mdtq)
        ...
#prg principal
Mamedia =[]
Choix = '1'
While choix != '0' :
        print(« Menu: »)
        print(« 0. Quitter »)
        print(« 1. Ajouter un élément »)
        print(« 2. Afficher contenu médiatique »)
        print(« 3. Rechercher des éléments »)
        print(« 4. Afficher statistiques »)
        choix = input (« votre choix ? »)
        if choix == '1' :
                item = saisie_item()
                MaMedai.add(item)
        if choix == '2 :
                afficherM(MaMedia)
        ...
```

REPRISE

Login Polytech
Identifiant : UNIV-TOURS\2025578t
(votre numéro étudiant suivi de la lettre t)
Mot de Passe : etu2015ST

Petit bilan

Une application Python typique sera alors constituée

- d'un programme principal,
- de un ou plusieurs modules contenant chacun les définitions d'un certain nombre de fonctions accessoires.

Résumé : structure d'un programme Python type

```
# -*- coding:Utf8 -*-

#####################################
# Programme Python type             #
# auteur : G.Swinnen, Liège, 2009   #
# licence : GPL                     #
#####################################


#####################################
# Importation de fonctions externes :

from math import sqrt


#####################################
# Définition locale de fonctions :

def occurrences(car, ch):
    "Cette fonction renvoie le \
     nombre de caractères <car> \
     contenus dans la chaîne <ch>"

    nc = 0

    i = 0

    while i < len(ch):

        if ch[i] == car:
            nc = nc + 1

        i = i + 1

    return nc


#################################
# Corps principal du programme :

print("Veuillez entrer un nombre :")
nbr = eval(input())

print("Veuillez entrer une phrase :")
phr = input()
print("Entrez le caractère à compter :")
cch = input()

no = occurrences(cch, phr)
rc = sqrt(nbr**3)

print("La racine carrée du cube", end=' ')
print("du nombre fourni vaut", end=' ')
print(rc)

print("La phrase contient", end=' ')
print(no, "caractères", cch)
```

Un programme Python contient en général les blocs suivants, dans l'ordre :
- Quelques instructions d'initialisation (importation de fonctions et/ou de classes, définition éventuelle de variables globales).
- Les définitions locales de fonctions et/ou de classes.
- Le corps principal du programme.

Le programme peut utiliser un nombre quelconque de fonctions, lesquelles sont définies localement ou importées depuis des modules externes.
Vous pouvez vous-même définir de tels modules.

La définition d'une fonction comporte souvent une liste de PARAMÈTRES.
Ce sont toujours des VARIABLES, qui recevront leur valeur lorsque la fonction sera appelée.

Une boucle de répétition de type 'while' doit toujours inclure au moins quatre éléments :
- l'initialisation d'une variable 'compteur' ;
- l'instruction while proprement dite, dans laquelle on exprime la condition de répétition des instructions qui suivent ;
- le bloc d'instructions à répéter ;
- une instruction d'incrémentation du compteur.

La fonction "renvoie" toujours une valeur bien déterminée au programme appelant.
Si l'instruction 'return' n'est pas utilisée, ou si elle est utilisée sans argument, la fonction renvoie un objet vide : 'None'.

Le programme qui fait appel à une fonction lui transmet d'habitude une série d'ARGUMENTS, lesquels peuvent être des valeurs, des variables, ou même des expressions.

Manipulation des fichiers

- Les fichiers permettent le transfert de données stockées en mémoire vive (RAM) vers la mémoire de masse (disque dur, clé usb, …) pour un **stockage pérenne.**
- Les **fichiers permettent** :
  - De conserver les données après fermeture du programme
  - D'échanger des données entre programmes
- Le système de fichiers est géré par le **système d'exploitation** (Linux, Windows, …)
  - Les fichiers doivent pouvoir être identifiés ➔ nom
  - Les fichiers doivent pouvoir être localisés ➔ répertoire (directory)
- Lorsque les quantités de données deviennent très importantes, il devient nécessaire de structurer les relations entre ces données, on utilise alors des systèmes plus sophistiqués : les **bases de données relationnelles**
  - Oracle, Sybase, PostgreSQL, MySQL, …

Manipulation des fichiers

Importation du module « os » et changement de répertoires

- **Répertoire courant**

```
import os
rep_cour = os.getcwd()
print (rep_cour)
```
C:\Python\essais

- **Changement de répertoire courant**

```
import os
os.chdir('d://home//jy//exercices')
print ( os.getcwd() )
```
C:\home\jy\exercices

Manipulation des fichiers Texte

Ouverture et fermeture de Fichiers Texte

- L'accès aux fichiers est assuré par l'intermédiaire d'un objet particulier : **objet-fichier** créé par l'instruction

of = open(nomFichier, modeOuverture)

- ModeOuverture peut prendre les valeurs suivantes
  'r' ➔ read = Ouverture en lecture
  'a' ➔ append = Ouverture en écriture à la suite des données déjà présentes dans le fichier
  'w' ➔ write = Ouverture en écriture avec effacement de l'existant

Exemple:

```
obFichier = open('Monfichier','a')
.....
obFichier.close()    # Ne pas oublier de fermer le fichier après utilisation
```

Manipulation des fichiers Texte

Lecture (read) et écriture (write) dans des Fichiers Texte

Exemple (écriture) :

```
obFichier = open('Monfichier','a')
obFichier.write('Bonjour, fichier ! \n')
obFichier.write("Quel beau temps,\n aujourd'hui ! \n")
obFichier.close()
```

Exemple (lecture) :

```
ofi = open('Monfichier', 'r')
t = ofi.read()          #la totalité du fichier est transféré dans la variable t
print(t)                # de type string
t = ofi.read(7)         # Nombre de caractères à lire
t = ofi.readline()      # Lecture d'une ligne jusqu'au caractère \n
lst = ofi.readlines()   # Lecture des lignes et stockage dans une liste
```

Manipulation des fichiers Texte

Lecture (read) et écriture (write) dans des Fichiers texte

Exemple plus complet :

```
def copieFichier(source, destination):
  #copie intégrale d'un fichier
  fs = open(source, 'r')
  fd = open(destination, 'w')
  while True :
      txt = fs.readline()
      if txt =="":
            break
       fd.write(txt)
  fs.close()
  fd.close()
```

Exercice : Faire une fonction qui recopie un script .py en supprimant les lignes de commentaire (commencant par#)

Manipulation des fichiers binaires

Ouverture et fermeture de Fichiers binaire

- Nécessite l'importation du module **pickle**
- Permet de stocker des données non textuelles (variables, listes, …)
- Mode d'ouverture et fermeture similaire aux fichiers Texte

of = open(nomFichier, modeOuverture)

- ModeOuverture peut prendre les valeurs suivantes
  'rb' ➔ read = Ouverture en lecture
  'ab' ➔ append = Ouverture en écriture à la suite des données déjà présentes dans le fichier
  'wb' ➔ write = Ouverture en écriture avec effacement de l'existant

Exemple:

```
obFichier = open('Monfichier',`rb')
obFichier.close()
```

Manipulation des fichiers

Lecture (load) et écriture (dump) dans des Fichiers binaire

```
import pickle
a, b, c = 27, 12.96, [5, 4.83, "René"]
f = open('donnees_test', 'wb')
pickle.dump(a, f)
pickle.dump(b, f)
pickle.dump(c, f)
f.close()

f = open('donnes_test', 'rb')
j = pickle.load(f)
k = pickle.load(f)
l = pickle.load(f)
print(j, type(j))
...
f.close()
```

→

```
27 <class 'int'>
12.96 <class 'float'>
[5, 4.83, 'Rene'] <class 'list'>
```

TD 4

- **Voir énoncé distribué**

TD 4 exercice 1

Correction exercice 1 TD4

```
import os

def compare(source1,source2):
```

86

Correction Ex 2

```
#Importation des modules utiles
Import  os

#Definition des fcts
def Affiche(nom) :
  fs = open(nom, 'r')
  print("===",nom,"====")
  while True:
      txt1 = fs.readline()
      if txt1 =="" :
                break
      print(txt1)
  fs.close()
  print("=======")

def Ajoute(nom) :
  fs = open(nom, 'a')
  ligne="?"
  while ligne != "":
      ligne =input("Ligne=?")
      if ligne != "":
          ligne = ligne + "\n"
          fs.write(ligne)

  fs.close()
```

```
#Prg principal
#init. variables
Nomf = input("Nom du fichier ?")
choix = "10"
os.chdir("d://exo")
#Boucle de choix
While choix != "0":
    print("MENU")
print("1. Afficher le contenu du fichier")
print("2. Ajouter du contenu dans le fichier")
    print("0. Quitter")
    choix = input("Votre choix ?")

#Realisation de l action demandée
    if choix == "1" :
        Affiche(Nomf)
    if choix == "2" :
        Ajoute(Nomf)
```

Introduction à la programmation avec Python - Partie 3

Interface Homme-Machine, Gestion des erreurs, …

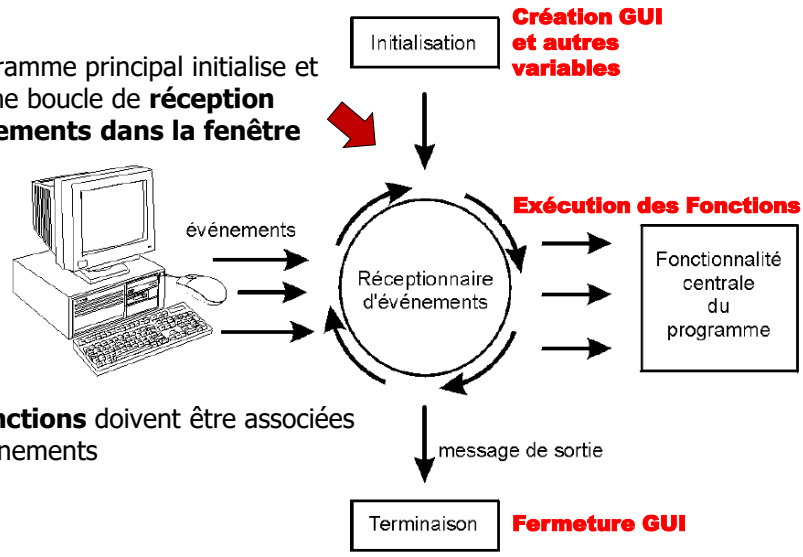
Du Texte aux Interfaces Graphiques…

- Jusqu'à présent, tous nos programmes ne fonctionnaient qu'en mode texte
- Affichage ➔ print() – Demande de saisie au clavier ➔ input()
- La mise au point d'applications évoluées exploitant correctement **des interfaces graphiques (ou GUI, Graphical User Interfaces)** est extrêmement complexe car :
  - Chaque système d'exploitation utilise des technologies / bibliothèques différentes (Windows, Linux, MacOS, …)
  - La programmation procédurale ne suffit plus, il faut passer à la **programmation orientée objet** (non étudiée dans ce cours)
- Il s'agira donc, dans ce cours, que d'un aperçu permettant de concevoir de petits programmes plus sympathiques

89

Du texte aux Interfaces Graphiques…

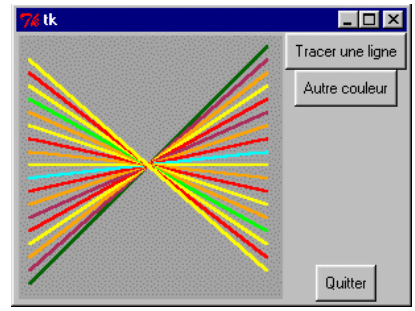
- Le programme principal initialise et lance une boucle de **réception d'événements dans la fenêtre**



- **Des fonctions** doivent être associées aux événements

Fenêtres & widgets

- Une **interface** est constituée d'une **fenêtre graphique** dans laquelle des **éléments d'interfaces** (**widgets** : objets graphiques) de différents types sont insérés
- A chaque widget sont associés :
  - Des propriétés et méthodes (fonctions) spécifiques permettant de le configurer
  - Des événements auxquels peuvent être associés des actions **(nos fonctions)**
- Principaux Widgets
  - Menu et sous menu (popup)
  - Boutons
  - Label (zones de texte)
  - Canevas (zone de dessin)
  - Entry (zone de saisie)
  -



Principaux widgets

- Utilisation du module python **tkinter**

Widget	Description
Button	Un bouton classique, à utiliser pour provoquer l'exécution d'une commande quelconque.
Canvas	Un espace pour disposer divers éléments graphiques. Ce widget peut être utilisé pour dessiner, créer des éditeurs graphiques, et aussi pour implémenter des widgets personnalisés.
Checkbutton	Une case à cocher qui peut prendre deux états distincts (la case est cochée ou non). Un clic sur ce widget provoque le changement d'état.
Entry	Un champ d'entrée, dans lequel l'utilisateur du programme pourra insérer un texte quelconque à partir du clavier.
Frame	Une surface rectangulaire dans la fenêtre, où l'on peut disposer d'autres widgets. Cette surface peut être colorée. Elle peut aussi être décorée d'une bordure.
Label	Un texte (ou libellé) quelconque (éventuellement une image).
Listbox	Une liste de choix proposés à l'utilisateur, généralement présentés dans une sorte de boîte. On peut également configurer la Listbox de telle manière qu'elle se comporte comme une série de « boutons radio » ou de cases à cocher.

92

¨Principaux widgets

- Suite…

Menu	Un menu. Ce peut être un menu déroulant attaché à la barre de titre, ou bien un menu « *pop up* » apparaissant n'importe où à la suite d'un clic.
Menubutton	Un bouton-menu, à utiliser pour implémenter des menus déroulants.
Message	Permet d'afficher un texte. Ce widget est une variante du widget Label, qui permet d'adapter automatiquement le texte affiché à une certaine taille ou à un certain rapport largeur/hauteur.
Radiobutton	Représente (par un point noir dans un petit cercle) une des valeurs d'une variable qui peut en posséder plusieurs. Cliquer sur un bouton radio donne la valeur correspondante à la variable, et « vide » tous les autres boutons radio associés à la même variable.
Scale	Vous permet de faire varier de manière très visuelle la valeur d'une variable, en déplaçant un curseur le long d'une règle.
Scrollbar	Ascenseur ou barre de défilement que vous pouvez utiliser en association avec les autres widgets : Canvas, Entry, Listbox, Text.
Text	Affichage de texte formaté. Permet aussi à l'utilisateur d'éditer le texte affiché. Des images peuvent également être insérées.
Toplevel	Une fenêtre affichée séparément, au premier plan.

Premier Exemple

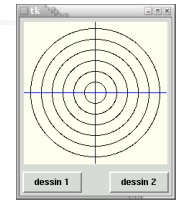
Définition d'une fenetre avec une zone de texte et un bouton

```
from tkinter import *              # idem import tkinter

fen1 = Tk()                        # Creation de la fenetre
tex1 = Label(fen1, text='Bonjour tout le monde !', fg='red')      #zone texte
tex1.pack()                        # insertion dans la fenetre
bou1 = Button(fen1, text='Quitter', command = fen1.quit)         #Bouton
bou1.pack()                        # insertion dans la fenetre
fen1.mainloop()                    # lancement boucle d'attente d'événements
fen1.destroy()                     # destruction fenetre
```

Les widgets peuvent etre vus comme des variables spéciales (objets) créées et configurées à l'aide d'appels de fonctions (de création/ modification)

Rem : fen1.quit() ➔ Sortie du mainloop()

Prise en main des widgets par l'exemple



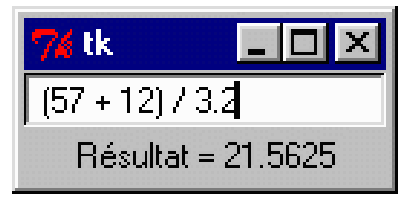
Reste maintenant à apprendre à créér et manipuler les multiples widgets ➔ Il faut lire les documentations et pratiquer ...

```
from tkinter import *
def cercle(x, y, r, coul ='black'):
"trace d'un cercle de centre (x,y) et de rayon r"
   can.create_oval(x-r, y-r, x+r, y+r,
                                  outline=coul)

def figure_1():
"dessiner une cible"
   # Effacer d'abord tout dessin préexistant :
   can.delete(ALL)
   # trace llignes :
   can.create_line(100, 0, 100, 200, fill
='blue')
   # tracer cercles concentriques :
    rayon = 15
    while rayon < 100:
         cercle(100, 100, rayon)
         rayon += 15
```

```
def figure_2():
"dessiner un cercle"
      can.delete(ALL)
      cercle(10, 20, 8,"red")

##### Programme principal : #####
fen = Tk()
can = Canvas(fen, width =200, height
=200, bg ='ivory')
can.pack(side =TOP, padx =5, pady =5)
b1 = Button(fen, text ='dessin 1',
command = figure_1)
b1.pack(side =LEFT, padx =3, pady =3)
b2 = Button(fen, text ='dessin 2',
command = figure_2)
b2.pack(side =RIGHT, padx =3, pady =3)
fen.mainloop()
```


Prise en main des widgets par l'exemple

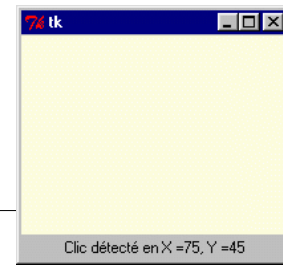
```
# Calculatrice graphique
from tkinter import *
from math import *

# définition de l'action à effectuer si l'utilisateur actionne
# la touche "enter" alors qu'il édite le champ d'entrée :
def evaluer(event):
     chaine.configure(text = "Résultat = " + str(eval(entree.get())))

# ----- Programme principal : -----
fenetre = Tk()
entree = Entry(fenetre)
entree.bind("<Return>", evaluer)
chaine = Label(fenetre)
entree.pack()
chaine.pack()
fenetre.mainloop()
```



Association Evenement ➔ Action

Prise en main des widgets par l'exemple

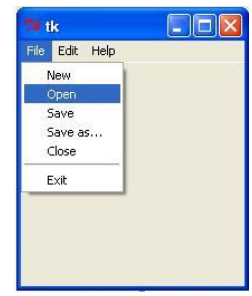
```
# Détection et positionnement d'un clic de souris dans une fenêtre :
from tkinter import *
def pointeur(event):
    chaine.configure(text = "Clic détecté en X =" + str(event.x) +\
     ", Y =" + str(event.y))

fen = Tk()
fen.title("Nom de la fenetre")
cadre = Frame(fen, width =200, height =150, bg="light yellow")
cadre.bind("<Button-1>", pointeur)
cadre.pack()
chaine = Label(fen)
chaine.pack()
fen.mainloop()
```



97

Mise en place de menus

```
root = Tk()
def hello():
    print 'hello!'
def popup():
    x,y = 11,12     #position du menu popup
    helpmenu.post(x, y)
menubar = Menu(root)
# create a pulldown menu, and add it to the menu bar
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Open", command=hello)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="File", menu=filemenu)
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="About", command=hello)
menubar.add_cascade(label="Help", menu=helpmenu)
# display the menu
root.config(menu=menubar)
```



Positionnement des widgets

- Chaque widget doit être positionné dans la fenêtre
- Trois méthodes différentes existent :
  - la méthode place()

    txt1.place(x = 20, y =30, width=120, height=25, relwidth=.2, relheight=.3, relx=.1, rely=.2, …)

    Positionnement absolue ➔

  - la méthode grid()

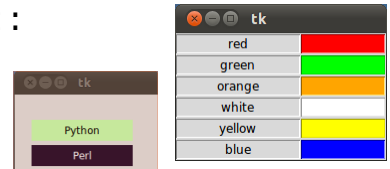
    txt.grid(row=0, column=2, columnspan=2, rowspan=2, sticky=W+E+N+S, padx=5, pady=5)

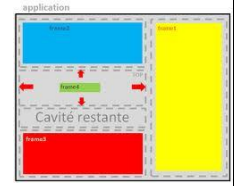
    Quadrillage de l'espace (tableau) ➔

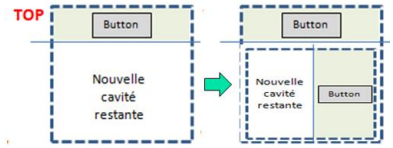
  - la méthode pack()

    txt1.pack(side =LEFT TOP BOTTOM RIGHT, fill=NONE X Y, expand=0 1)

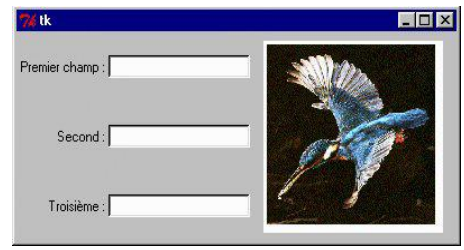
    Remplissage séquentielle des cavités restantes ➔








```
fen1 = Tk()
# création de widgets 'Label' et 'Entry' :
txt1 = Label(fen1, text ='Premier champ :')
txt2 = Label(fen1, text ='Second :')
txt3 = Label(fen1, text ='Troisième :')
entr1 = Entry(fen1)
entr2 = Entry(fen1)
entr3 = Entry(fen1)
```




```
# création d'un widget 'Canvas' contenant une image bitmap :
can1 = Canvas(fen1, width =160, height =160, bg ='white')
photo = PhotoImage(file ='martin_p.gif')
item = can1.create_image(80, 80, image =photo)

# Mise en page à l'aide de la méthode 'grid'
txt1.grid(row =1, sticky =E)
txt2.grid(row =2, sticky =E)
txt3.grid(row =3, sticky =E)
entr1.grid(row =1, column =2)
entr2.grid(row =2, column =2)
entr3.grid(row =3, column =2)
can1.grid(row =1, column =3, rowspan =3, padx =10, pady =5)

# démarrage :
fen1.mainloop()
```

Position dans le canvas

Etalement sur 3 colonnes

Espacement en x et y

Gestion des erreurs

- Mécanisme de traitement des Exceptions :
  - Les exceptions sont les opérations qu'effectuent un interpréteur lorsqu'une erreur est détectée au cours de l'exécution d'un programme
  - L'exécution du programme est interrompue et un message d'erreur plus ou moins explicite est affiché
  - Exemple :

```
>>> print(55/0)
ZeroDivisionError: int division or modulo by zero
```

  - Le message d'erreur comporte deux parties séparées par ':'
  - Type d'erreur : une information spécifique sur cette erreur

  - il est possible de prévoir à l'avance certaines des erreurs qui risquent de se produire et d'inclure à cet endroit des instructions particulières qui seront activées seulement si ces erreurs se produisent

---

Gestion des erreurs

- Mécanisme de traitement des Exceptions :

  - **L'ensemble d'instructions try - except – else permet d'intercepter une erreur et d'exécuter une portion de script spécifique pour cette erreur**

  - Le bloc d'instructions qui suit directement une instruction **try** est exécuté par Python sous réserve d'erreur
  - Si une erreur survient pendant l'exécution de l'une de ces instructions, alors Python annule cette instruction fautive et exécute à sa place le code inclus dans le bloc qui suit l'instruction **except**
  - Si aucune erreur ne s'est produite dans les instructions qui suivent **try**, alors c'est le bloc qui suit l'instruction **else** qui est exécuté (si cette instruction est présente).
  - Dans tous les cas, l'exécution du programme peut se poursuivre ensuite avec les instructions ultérieures

---

Gestion des erreurs

- **Mise en oeuvre :**

try :
    #Bloc à essayer/executer
exept *Nom de l'exception* **:**
    #Bloc à executer en cas d'erreur
exept *Nom de l'exception* **:**
    #Bloc à executer en cas d'erreur
else:
    #Bloc à executer si pas d'erreur
finally:
    #Bloc a executer dans tous les cas

#suite du programme

Optionnel

Exemple de nom d'exception :
ZeroDivisionError

Exemple

```
fnm = input("Veuillez entrer
                    un nom de fichier : ")
try:
f = open(fnm, "r")
except:
print("Le fichier", fnm, "est introuvable")
```