PYTHON

LES BASES EN UNE HEURE

Sources et ressources

- Sources d'inspiration
  - Cours de Laurent Pointal (https://perso.limsi.fr/pointal)
  - Cours de Charles Severance (http://open.umich.edu/education/si/coursera-programming-everybody)
  - Cours réseau de Jean-Yves Thibon (http://igm.univ-mlv.fr/~jyt/python/python_5.pdf)
- Ressources numériques
  - LA Math physique Chimie
  - L2 Math "Algorithmique & Programmation_ http://umtice.univ-lemans.fr/course/view.php?id=3026

Python

- Avantages
  - Extrêmement portable: Unix, Windows, Mac OS, Android, iOS, systèmes embarqués
  - Sûr : pas de pointeurs, gestion automatique de la mémoire
  - Nombreuses bibliothèques : réseau, bases de données, interface graphique, vidéo, calcul scientifique, web…
  - Extensible C / C++ / Fortran…
- Historique
  - Première version : février 1991
  - Version 2.7 et version 3.5
    - quelques incompatibilités entre les deux

Python : Caractéristiques

- Tout est un objet
- Modules, classes, fonctions
- Exceptions
- Typage dynamique, polymorphisme
- Surcharge des opérateurs
- Syntaxe simple, compacte et intuitive
  - Mélange de C / C++
  - Indentation pour délimiter les blocs
- Multi paradigme : impératif, objet, fonctionnel ...
- Script ou ligne de commande

Syntaxe : comparaison

- Java :

```
public class Hello {
  public static void main(String[] args) {
       System.out.println("Hello, World!");
  }
}
```

- C

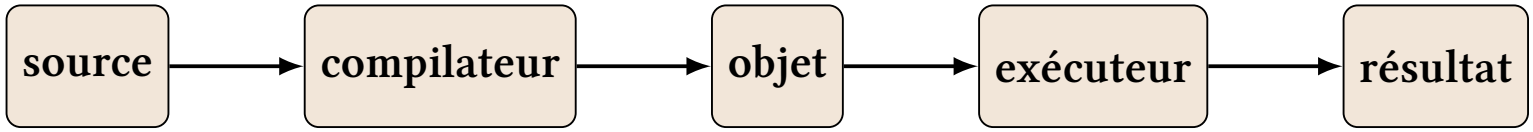
```
Void main(void) {
  printf("Hello, World!");
}
```

- Python :

```
print("Hello, World!")
```

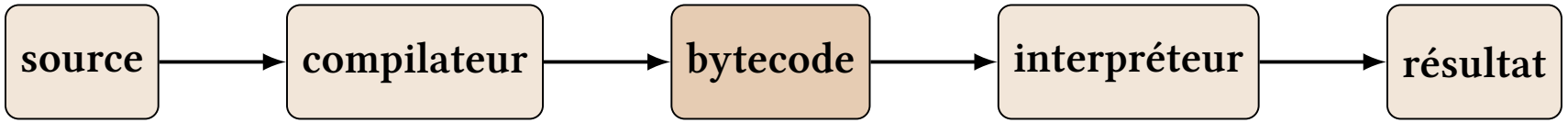
Python

- Code compilé



- Code interprété

- Python = bytecode compilé qui est interprété
  - version : Python 2.7 vs Python 3



Type simple et constante

- Entier : int
  - `-1` `0` `1532`
- Décimaux : float
  - `1.3` `-67.67` `12.5e-9`
- Booléen : bool
  - `True` `False`
- Rien, indéfini
  - `None`

Type conteneur : liste, str, tuple et dict

- Liste de valeurs : `list()`
  - `[1, 2, 3, 4]`
  - `[1, 2.0]`
  - Chaîne de caractères : `str()`
    - `'azerty' "azerty"`
    - = **une liste de caractères non modifiable**
  - Tuple : liste de valeurs non modifiables
    - `(1, 2, 3)`
- Dictionnaire = tableau associatif : `dict()`
  - Associer à une clé une valeur
  - `{'money': 12, 'tissues': 75, 'candy': 3}`
  - `{1: 120, 2: 'azerty'}`

- Attention :
  - liste, str et dict sont des objets qui possèdent des méthodes
  - Passage et affectation par référence

Constante et variable

- Pas de constantes littérales, que des variables
- Variable : typée, pas de déclaration, le typage est dynamique
- **Toute variable a une valeur et un type**
- Conversion
  - int()
    - `int('123')` `int(1.2)` `int('azert')`
  - float()
    - `float('123.0')` `float(1)` `float('azerty')`
  - str()
    - `str(123.0)` `str(1)` `str('azerty')`
  - bool()
    - `bool(1)` `bool(0)` `bool('azerty')`
- Identifiant (nom de variable ou de fonction) : [_a-zA-Z][_a-zA-Z0-9]*
  - Mots séparés par _

Opérateurs, priorité

- Nouveau
  - Puissance, Division, Concaténation de chaînes
  - + * applicable aux listes
- Priorité la plus haute à la plus basse
  - Les parenthèses sont toujours respectées
  - L'exposant (élévation à la puissance)
  - Multiplication, division et modulo (reste)
  - Addition et soustraction
  - De gauche à droite

Opérateur	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division
//	Division entière
**	Puissance
%	Modulo

Parenthèses
Puissances
Multiplications & div
Additions & sous
De gauche à droite

Opérations logiques

- Valeurs booléennes
  - Vrai : `True` = 1
  - Faux : `False` = 0
- Opérateurs booléens
  - Et → `and`, ou → `or`, non → `not`
- Opérateurs de comparaison
  - Idem au C
  - Applicable aux listes, chaînes et dictionnaires

Opérateur binaire	Signification
`<`	Inférieur à
`<=`	Inférieur ou égal à
`==`	Egal à
`>=`	Supérieur ou égal à
`>`	Plus grand que
`!=`	Différent de

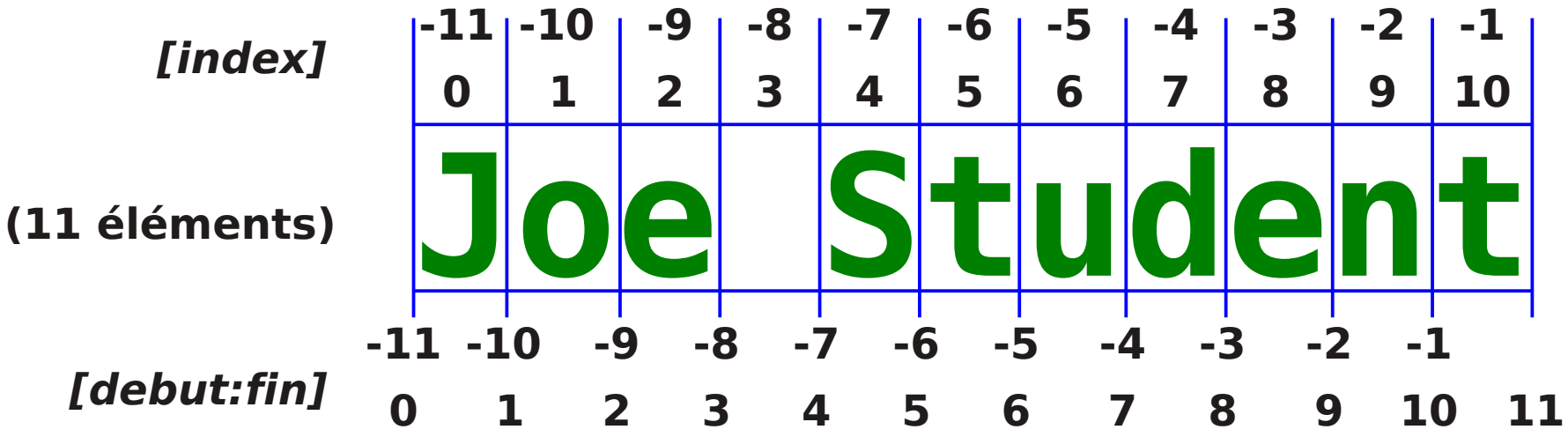
Opérateur unaire	Signification
`not`	négation

Opérateur d'affection

- Opérateur `=`
- Ex : `a = 3*2`
- En 3 étapes
  - 1- évaluer la partie de droite `3*2`
  - 2- création de la variable `a`, si elle n'existait pas
  - 3- association entre le nom de la variable et la valeur
- Affectation multiple
  - `a, b = 1 , 2`
  - `a, b = b, a`
- Opération puis affectation
  - [opérateur]=
  - Opérateur in [+, -, *, **, /, //, %]

Indexation

- Pour les listes et les chaînes de caractères
  - Accès à un élément : `a[index]`
- Extraire une sous séquence (slice = tranche)
  - `a[start:stop]`
  - `a[start:stop:step]`



Entrées / Sorties console

- Saisir : `input()` retourne une chaîne de caractères

```
a = input()
a = input('saisir un mot')
a = int(input('saisir un entier'))
```

- Afficher : `print()`

```
print('bonjour')
print(a, b, c)
```

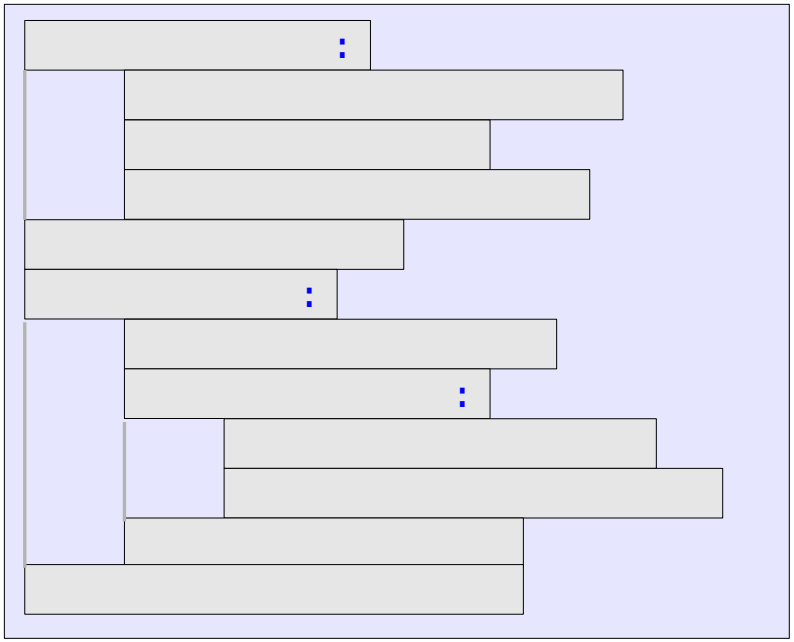
Flot d'instruction

```
##################################################
# Exercice 6
##################################################

ANNEE = 2015
nom = input("Quel est ton nom : ")
prenom = input("Quel est ton prenom : ")
annee = int(input("Quelle est ton année de naissance : "))
age = ANNEE - annee
print("Bonjour", prenom, nom, "tu es âgé de", age, "ans")
print("Bonjour",prénom, nom,"tu es agé de ", age," ans")
```

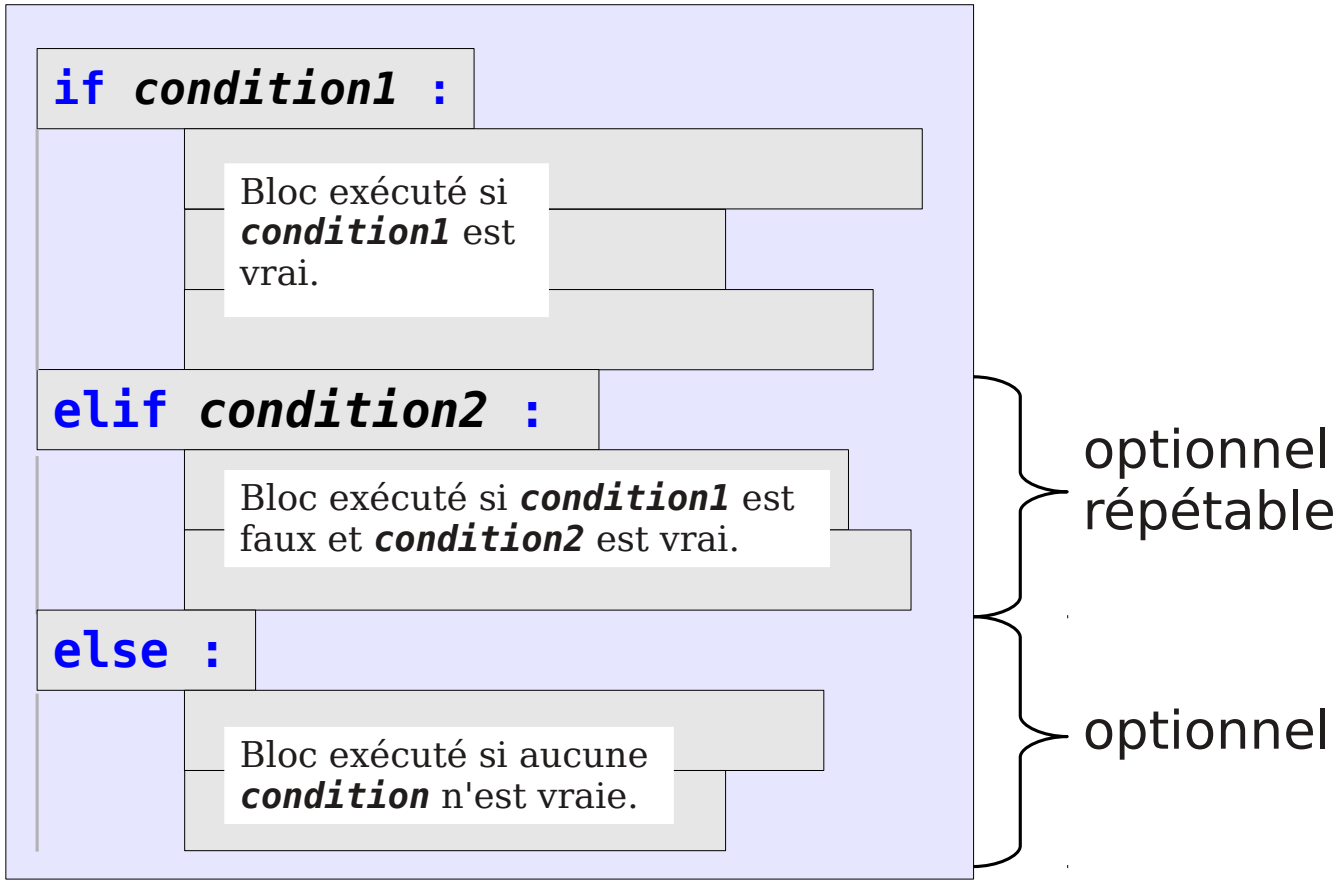
Blocs d'instruction

- Attention l'indentation détermine les blocs !
  - Entête du bloc se termine par :
  - Les instructions du bloc indentées par rapport à l'entête (généralement 4 espaces)
  - Retour au niveau d'indentation qui précède marque la fin du bloc
  - Les lignes vides sont sans effet



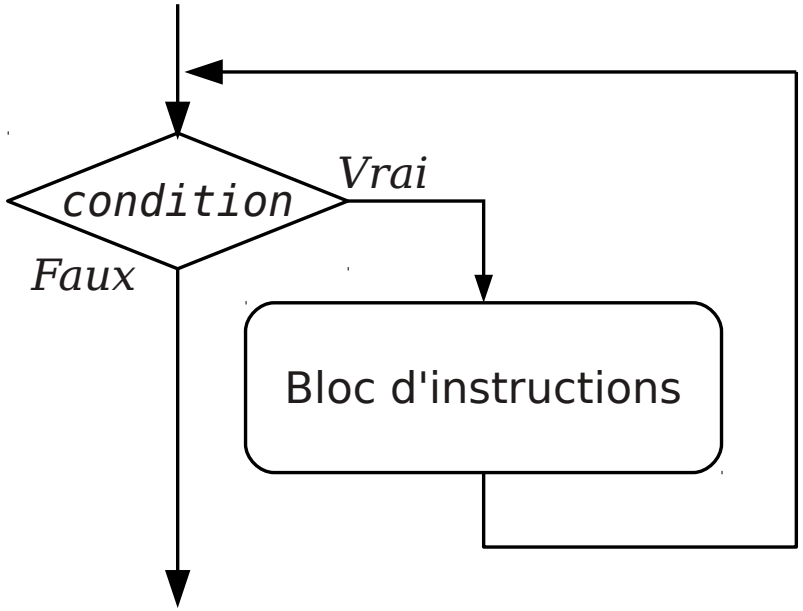
Instruction conditionnelle

- Alternative entre plusieurs blocs en fonction des valeurs d'expression



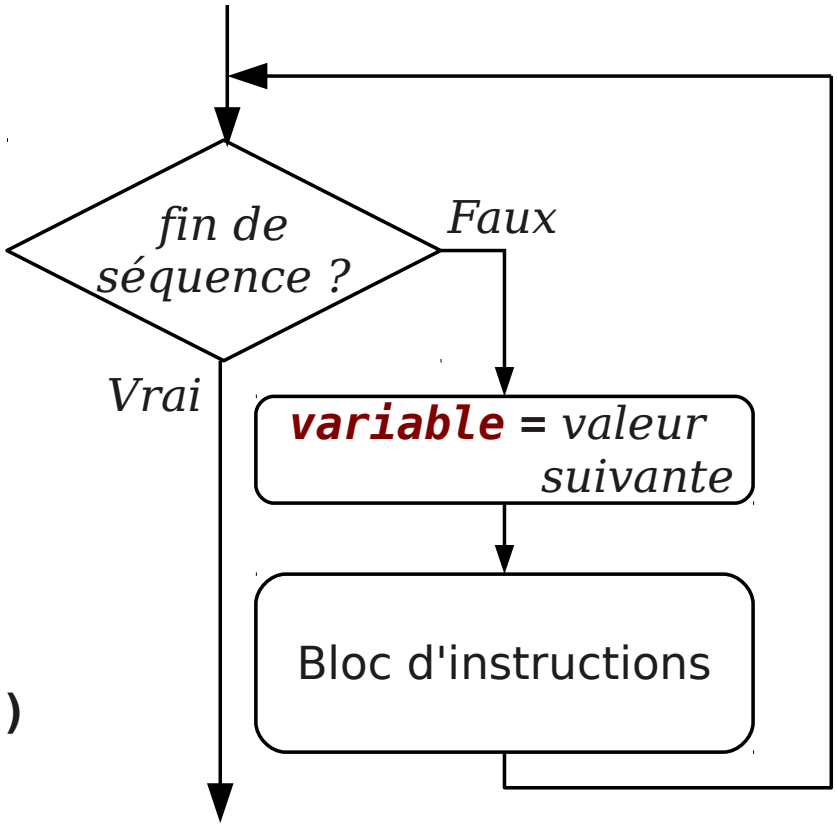
Instruction boucle while

```
while condition :
    Bloc d'instructions.
    Exécuté tant que la
    condition est vraie.
```



Instruction : parcours de séquence

```
for variable in séquence:
    Bloc d'instructions.
    Exécuté avec variable valant
    tour à tour chacune des
    valeurs de séquence.
```




```
valeurs = [ 1,4,2,8,9,5,7,6 ]
for v in valeurs:
    print(v,"\t==> carré:",v**2)
    print("\t==> cube:",v**3)
```

Fonctions utiles

- Type d'une variable : `type()`
- Obtenir des informations sur une variable : `dir(a)`
- Générer une liste d'entier

```
range(start, stop (non inclus), step)
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(10,20,2))
[10, 12, 14, 16, 18]
```

Méthodes : liste

- Créer une liste vide : `a = list()`
- Ajouter un élément : `a.append('azerty')`
- Concaténer deux listes : `a = a + [1, 2, 3]`
- Nombre d'éléments d'une liste : `n = len(a)`
- Insertion d'un élément : `a.insert(i, x)`
- Indexe de la 1er valeur x : `a.index(x)`
- Suppression d'une valeur : `a.remove(x)`
- Suppression du ième élément : `del a[i]`
- Tri de la liste : `a.sort()`
- Copie de la liste : `b = a.copy()`
- + `sum(), max(), min()`

Méthodes : chaîne de caractères

- Conversion en minuscule : `s.lower()`
- Conversion en majuscule: `s.upper()`
- Retirer les caractères [] en début et fin : `s.strip([chars])`
- Découper en mots : `s.split(' ')`
  - retourne une liste de chaînes de caractères correspondant aux différentes parties de la chaîne initiale coupée par la chaîne passée en argument
- Chercher : `s.find(str)`
  - retourne l'indice de la première occurrence de str
- Remplacer : `s.replace(str1, str2)`
  - retourne la chaîne de caractères avec toutes les occurrences de str1 remplacée par str2
- Compter: `s.count()`
  - retourne le nombre d'occurrences str
- `s.decode('utf8') s.encode('utf8')`

Méthodes : dictionnaire

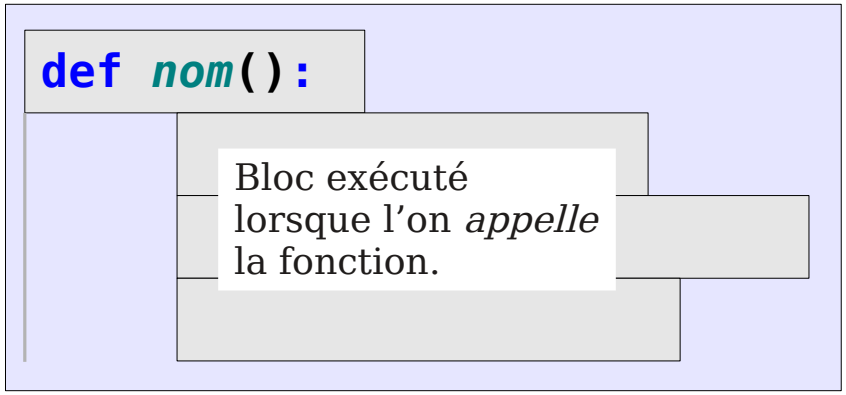
- Présence d'une clé : `key in d`
- Accès à une valeur : `d[key]`
- Affecter une valeur : `d[key] = value`
- Supprimer une entrée : `del d[key]`
- Liste des clés : `d.keys()`
- Liste des valeurs : `d.values()`
- Copier un dictionnaire : `d.copy()`

Objet et référence

- Les conteneurs sont des objets
  - un objet est un conteneur symbolique et autonome qui contient des attributs et des méthodes
  - proche d'une structure C contenant des pointeurs sur fonctions et des attributs.
- Les variables sont des références à des objets
  - Si `b = a`, une modification de `a` affecte `b`
- Copier un objet
  - Utiliser `copy.copy()` et `copy.deep_copy()` du module `copy`

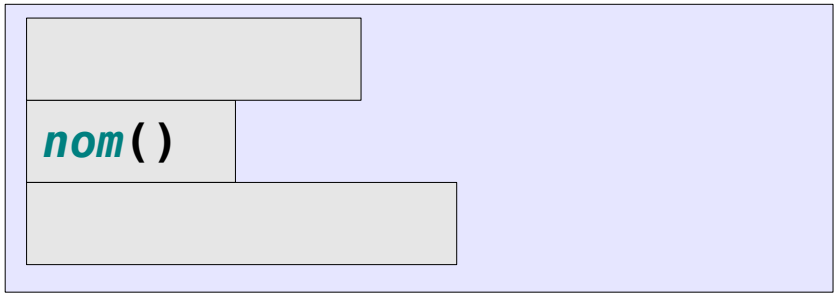
Fonction

- Un nom associé à un bloc d'instruction
- Définition




```
def fct():
    print("=" * 64)
    print(" ALERTE " * 8)
    print("=" * 64)
```

- Utilisation

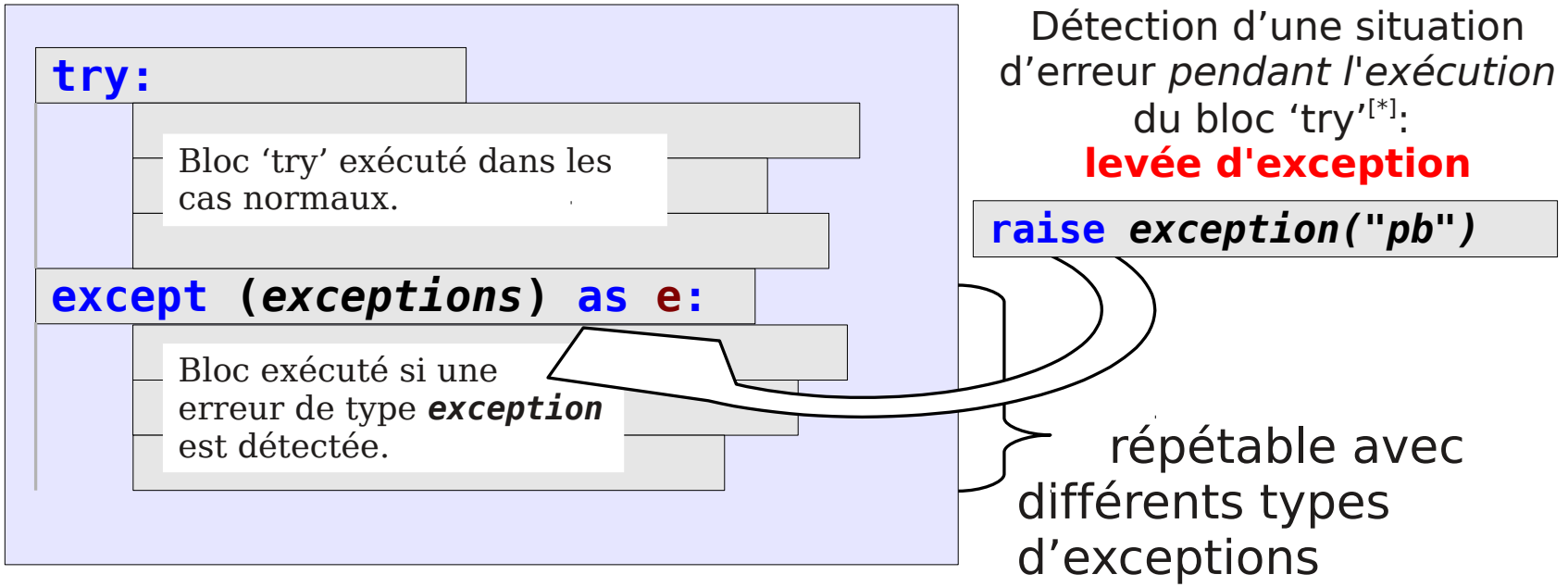


Fonction

- Paramètre d'une fonction
  - `def truc(a, b):`
- Valeurs de retour : return
  - `return a`
  - `return a, b`
- À chaque appel, il y a création des variables
- Portée et masquage des variables identiques au C
- Attention : les variables déclarées dans le bloc de niveau 1 sont des variables globales !

Exception

- Gestion des exception : `try / except`, où il faut :
  - Identifier la portion de code où l'exception peut survenir (être levée)
  - Écrire le code à exécuter s'il n'y a pas d'exception (traitement normal)
  - Écrire le code à exécuter en cas d'exception (traitement exceptionnel)



Main

```
b = 1

def main():     a = 10
    print(a, b)

if __name__ == "__main__":
    # execute only if run as a script
    main()
```

Modules

- Module = une bibliothèque de fonction

```
#!/bin/env python3
# -*- coding: utf-8 -*-
# Auteur: Joe Student
# Fichier: mon_premier_m.py
"""Calculs de nombres premiers.
"""
# Modules outils utilisés
import math,sys
import os

# Constantes et variables globales définies
LIMITE = 10000  # Ne pas dépasser lors des calculs.

# Définition des fonctions
def est_premier(p_n) :
    """Test si p_n est un nombre premier.
    """
    …
```

Modules

- Le stockage des modules dépend de la distribution python
  - `print(sys.path)`
  - cf PYTHONPATH
- Importer un module
  - `import nom_du_module`
- Importer certaines définitions d'un module
  - `from nom_du_module import fonction1, fonction2`
- Renommer
  - `import nom_du_module as m1`
  - `from nom_du_module import fonction1 as f1, fonction2`

Modules standards

- Accès à l'environnement, au répertoire courant, liste de fichiers, attributs des fichiers → `os`
  - Manipulations de chemins et de noms de fichiers → `os.path`
- Renommage, déplacement, copie, suppression de fichiers → `shutil`
- Fichiers temporaires → `tempfile`
- Fonction mathématique → `math`

Fichiers textes

- Ouverture d'un fichier texte

```
f = open("nomFichier","modeAccès",encoding="utf8")
```

Compréhension des accents des *fichiers textes*.

- **"r"** read (lecture, par défaut)
- **"w"** write (écriture, création fichier neuf vide)
- **"a"** append (ajout à la fin)

Et en plus

- **"t"** texte (par défaut)
- **"b"** binaire (sans interprétation des *fin de ligne*)
- **"+"** modification (lecture+écriture)

Chemin d'accès au fichier

Fichiers textes

Ouverture

```
f = open("toto.txt","r",encoding="utf-8")
f = open("toto.txt","w",encoding="utf-8")
```

Lecture

```
s = f.read()       # tout le fichier dans s
s = f.read(25)     # 25 caractères suivants dans s
s = f.readline()   # ligne suivante dans s
l = f.readlines() # toutes les lignes dans l
```

Écriture

```
f.write(s)        # chaîne s
f.writelines(l)   # liste de chaînes l
```

Fermeture

```
f.close()
```

Fichiers textes

- Boucles sur un fichier

```
f = open("data.txt",encoding="utf-8")
for line in f :
    # Traitement de la ligne
    print(line)
f.close()
```