Informatique et Programmation

Cours 17

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-py

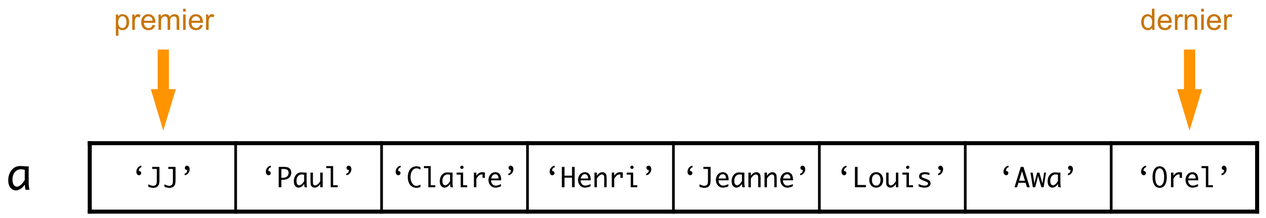
Plan

- files
- piles
- re-cap
- autres problèmes

dès maintenant: télécharger Python 3 en `http://www.python.org`

Files d'attente

- une file d'attente (*FIFO — First In First Out*)




```
def ajouter_file (x, a) :
    a.append(x)


def enlever_file () :
    try:
      del a[0]
    except Exception:
      print ('erreur')
```

```
def nouvelle_file () :
    return [ ]

a = nouvelle_file ()
```

les sources sont en http://jeanjacqueslevy.net/prog-py-22/progs/c7a.py

Files d'attente

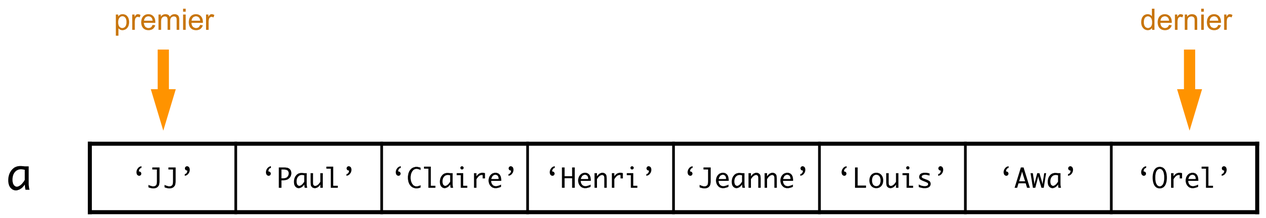
- représentation par une classe et objets


```
class FIFO :
    def __init__ (self, l) :
        self.liste = l
        self.premier = 0
        self.dernier = len(l) - 1
    #
    def __str__ (self) :
        return "FIFO ({})".format (self.liste)
    #
    def ajouter (self, v) :
        self.liste = self.liste + [v]
        self.dernier = self.dernier + 1
    #
    def retirer (self) :
        r = self.liste[0]
        del self.liste[0]
        return r
    #
    def __add__ (self, f) :
        return FIFO (self.liste + f.liste)
    #
    def est_vide (self) :
        return len(self.liste) == 0
```

les sources sont en http://jeanjacqueslevy.net/prog-py-22/progs/c7a.py

Files d'attente

• implémentations possibles



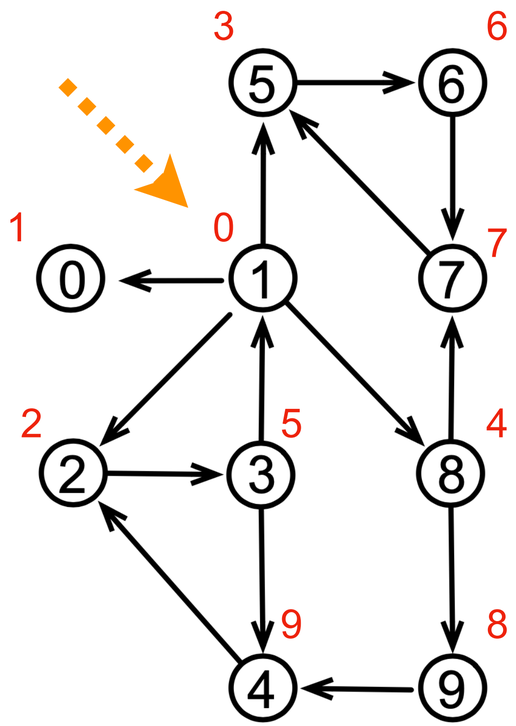
- de vraies listes (comme en Lisp, Ocaml, Haskell)

- tampon circulaire (comme en hardware)

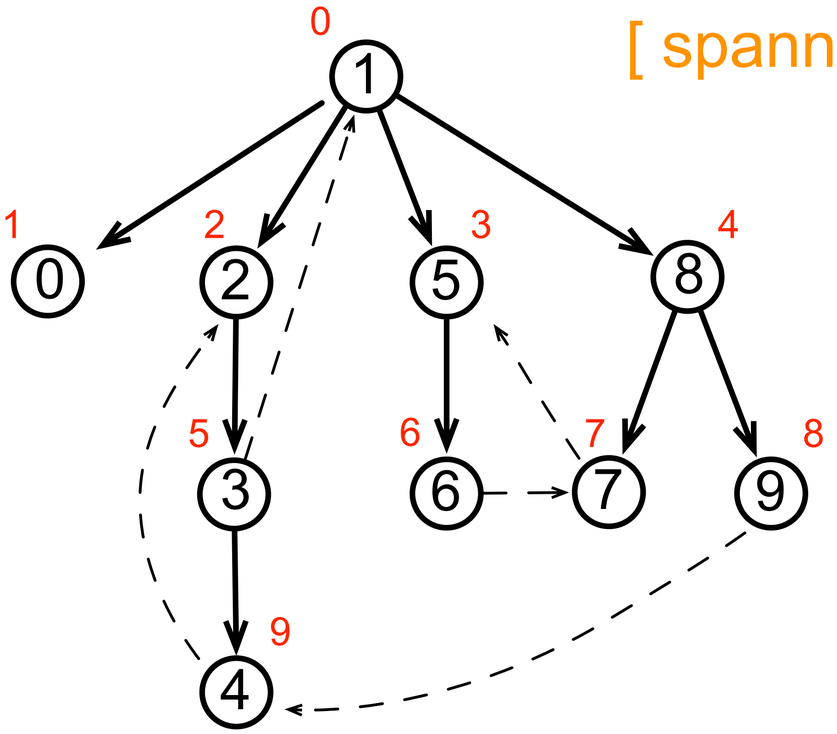
les sources sont en http://jeanjacqueslevy.net/prog-py-22/progs/c7a.py

BFS

- parcours en largeur d'abord (*breadth first search — BFS*)



arbre de recouvrement
[spanning tree]



- début du parcours en 1

- sur l'arbre de recouvrement, l'ordre de parcours est l'ordre militaire (selon la distance à partir de la racine)

BFS

- parcours en largeur d'abord (*breadth first search — BFS*)

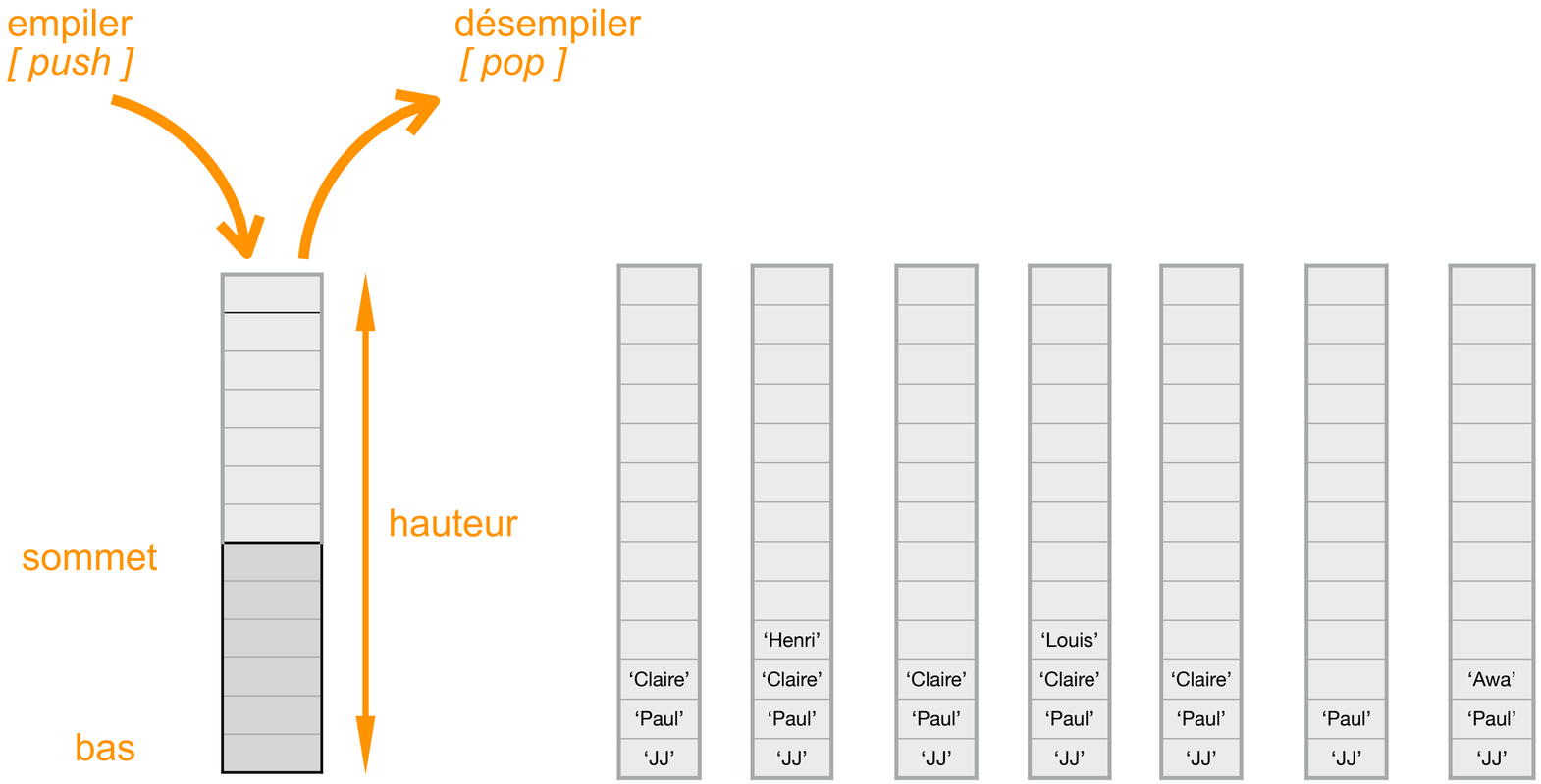
```
BLANC = 0; GRIS = 1; NOIR = 2
numOrdre = 0

def BFSde (g, x, num, couleur, f) :
    global numOrdre
    couleur[x] = GRIS; f.ajouter (x)
    while not f.est_vide() :
        x = f.retirer()
        num[x] = numOrdre; numOrdre += 1
        for y in g[x].voisins :
            if couleur[y] == BLANC :
                couleur[y] = GRIS; f.ajouter(y)
        couleur[x] = NOIR
    return num
```

```
def BFS (g) :
    global numOrdre
    n = len(g)
    num = n*[0]; couleur = n*[BLANC]
    numOrdre = 0
    f = FIFO ([ ])
    for x in range(n) :
        if couleur[x] == BLANC :
            BFSde (g, x, num, couleur, f)
    return num
```

les sources sont en http://jeanjacqueslevy.net/prog-py-22/progs/c7a.py

Piles

- une pile (*pushdown stack*) suit la stratégie *LIFO — Last In First Out*)



Piles

- représentation des piles par des listes

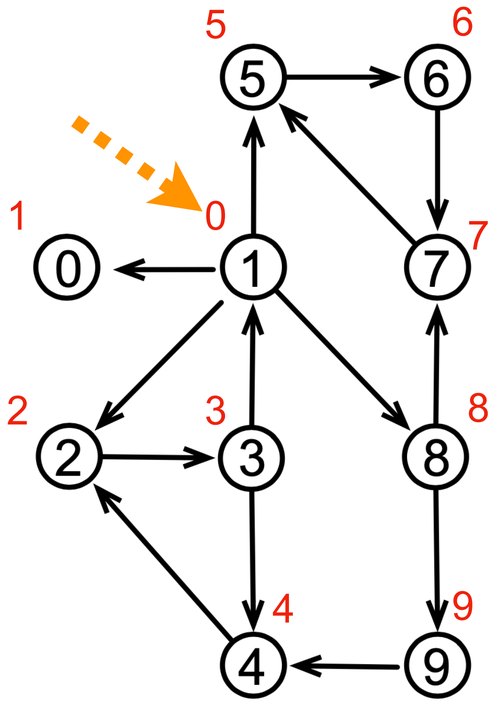
```
class Pile :
    def __init__ (self, l) :
        self.contenu = l
        self.contenu.reverse()
    #
    def __str__ (self) :
        return '{}'.format (self.contenu)
    #
    def empiler (self, x) :
        self.contenu = [x] + self.contenu
    #
    def desempiler (self) :
        r = self.contenu[0]
        del self.contenu[0]
        return r
    #
    def est_vide (self) :
        return len(self.contenu) == 0
```

```
p = Pile ([2, 3, 5])
print (p)
```

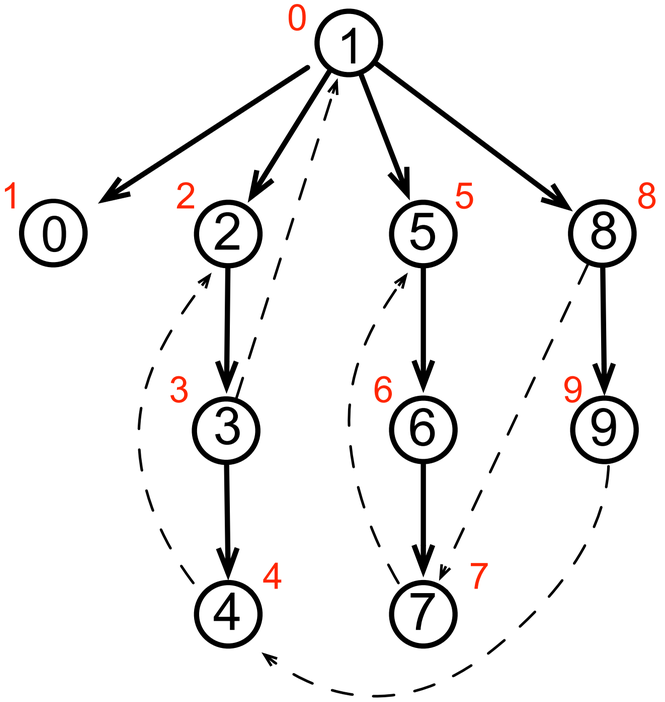
```
[5, 3, 2]
```

DFS

- parcours en **profondeur d'abord** (*depth first search — DFS*)



arbre de recouvrement
[spanning tree]



- début du parcours en 1

- les numéros du parcours en profondeur d'abord sont les numéros de l'ordre préfixe sur l'arbre de recouvrement

DFS

- parcours en profondeur d'abord (*depth first search — DFS*)

```
BLANC = 0; GRIS = 1; NOIR = 2
numOrdre = 0

def DFSde (g, x, num, couleur, p) :
    global numOrdre
    couleur[x] = GRIS; p.empiler (x)
    while not p.est_vide() :
        x = p.desempiler()
        num[x] = numOrdre; numOrdre += 1
        for y in g[x].voisins :
            if couleur[y] == BLANC :
                couleur[y] = GRIS; p.empiler(y)
        couleur[x] = NOIR
    return num
```

```
def DFS (g) :
    global numOrdre
    n = len(g)
    num = n*[0]; couleur = n*[BLANC]
    numOrdre = 0
    p = Pile([])
    for x in range(n) :
        if couleur[x] == BLANC :
            DFSde (g, x, num, couleur, p)
    return num
```

- comparer au parcours BFS !! et à la fonction récursive sur les chemins

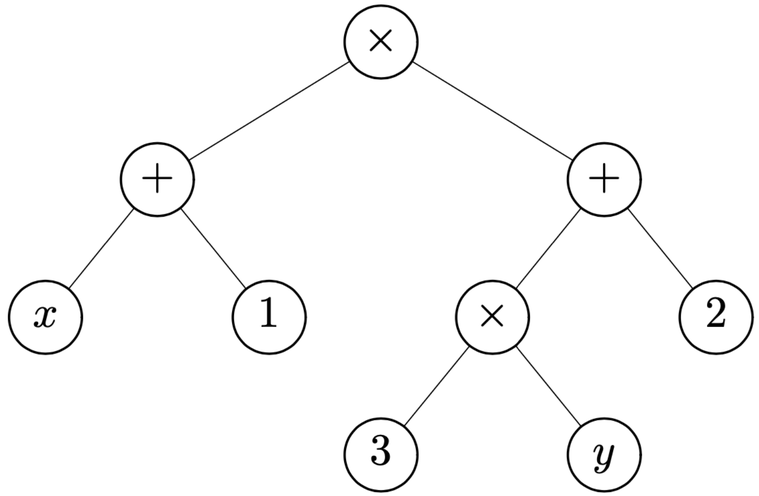
les sources sont en http://jeanjacqueslevy.net/prog-py-22/progs/c7a.py

AST (arbre de syntaxe abstraite)

- passer d'une chaîne de caractères à un arbre (syntaxe astraite) est plus difficile

$(x + 1) * (3 * y + 2)$

Analyse syntaxique



Belle impression

```
class Op_bi:
    def __init__ (self, x, g, d) :
        self.val = x
        self.gauche = g
        self.droit = d

class Op_un:
    def __init__ (self, x, a) :
        self.val = x
        self.fils = a

class CVar:
    def __init__ (self, x) :
        self.val = x
```

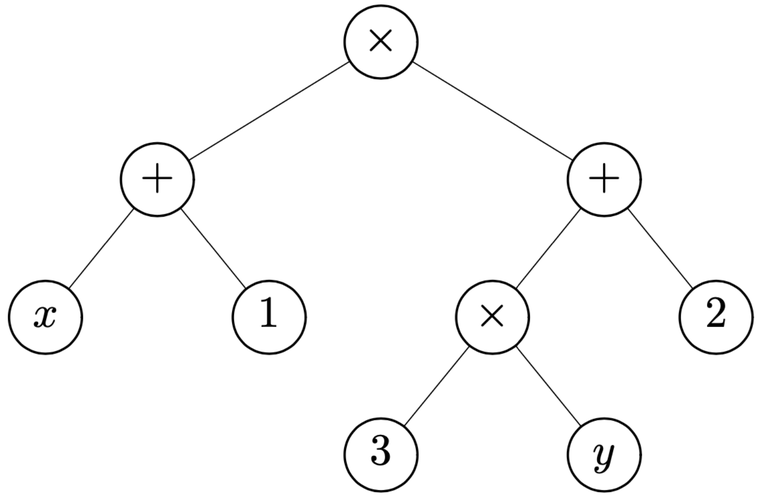
```
t = Op_bi ('*', Op_bi ('+', CVar ('x'), CVar (1)),
          Op_bi ('+', Op_bi ('*', CVar (3), CVar ('y')),
                      CVar (2)))
```

AST (arbre de syntaxe abstraite)

- on peut évaluer sa valeur en donnant une valeur aux variables *x* et *y*
- on définit l'environnement par le dictionnaire:

```
e = {'x' : 20, 'y' : -20}
```

```
def eval (t, e) :
    if isinstance (t, Op_bi) :
        if t.val == '+' :
            return eval (t.gauche, e) + eval (t.droit, e)
        elif t.val =='*' :
            return eval (t.gauche, e) * eval (t.droit, e)
    elif isinstance (t, Op_un) :
        return - eval (t.fils, e)
    elif isinstance (t.val, int) :
        return t.val
    else :
        return e[t.val]
```




```
t = Op_bi ('*', Op_bi ('+', CVar ('x'), CVar (1)),
          Op_bi ('+', Op_bi ('*', CVar (3), CVar ('y')),
                      CVar (2)))
```

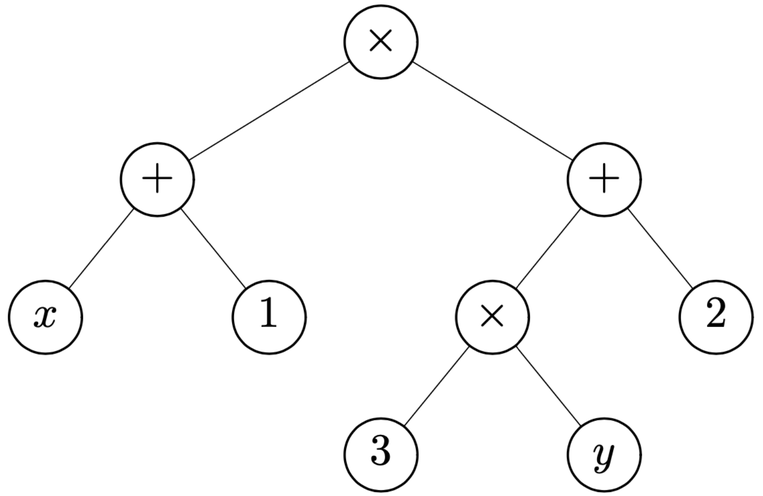
```
print (eval (t, e))
```

AST en notation polonaise

- générer les notations préfixe, postfixe et infixe

```
def polpostfix (a) :
    if isinstance (a, Feuille) :
        return a.val
    elif isinstance (a, Noeud_Un) :
        return polpostfix (a.fils) + ' ' + a.val
    else :
        return polpostfix (a.gauche) \
       + ' ' + polpostfix (a.droit) \
       + ' ' + a.val
```

- correspond au parcours **postfixe**



- notation polonaise postfixe

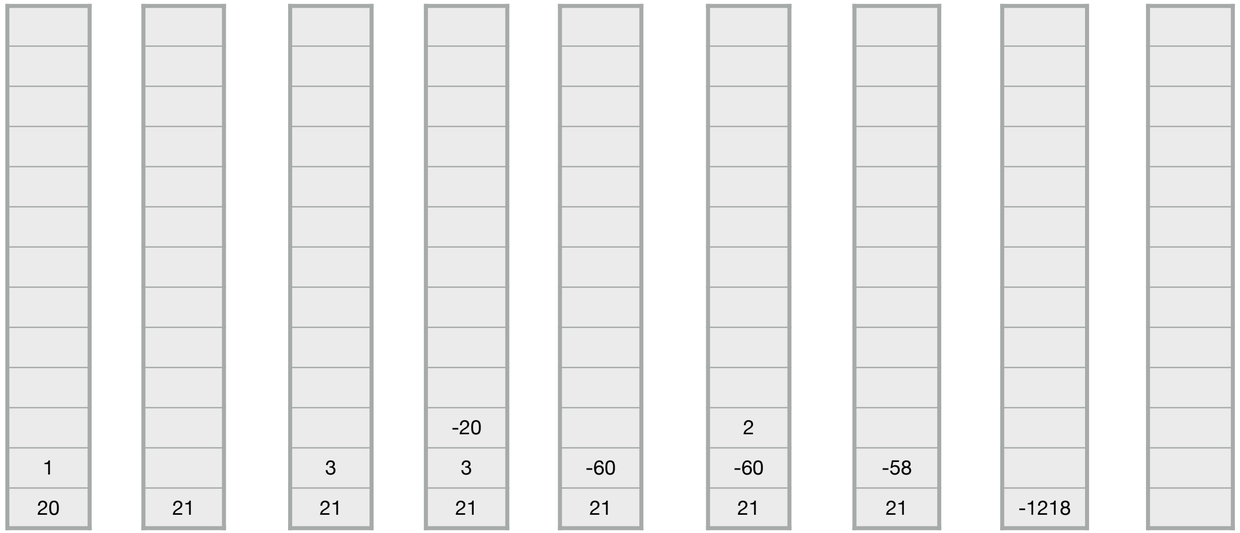
  x 1 + 3 y * 2 + *

AST - évaluation avec une pile

- on part de la notation polonaise (postfixe)

x 1 + 3 y * 2 + *

```
a = ['x', '1', '+', '3', 'y', '*', '2', '+', '*']

e = {'x' : 20, 'y' : -20}


def eval_pol_suffixe (a, e) :
    p = Pile ([ ])
    for t in a :
        if t == '+' :
            a1 = p.desempiler ()
            a2 = p.desempiler ()
            p.empiler (int(a1) + int(a2))
        elif t == '*' :
            a1 = p.desempiler ()
            a2 = p.desempiler ()
            p.empiler (int(a1) * int(a2))
        elif t in {'x', 'y'} :
            p.empiler (e[t])
        else :
            p.empiler (t)
    return p.desempiler()
```



Python - recap !

- mots clé déja vu

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

- différence entre `==` et `is`
  - `==` est l'égalité des valeurs
  - `is` est l'égalité des adresses mémoire (*references*)

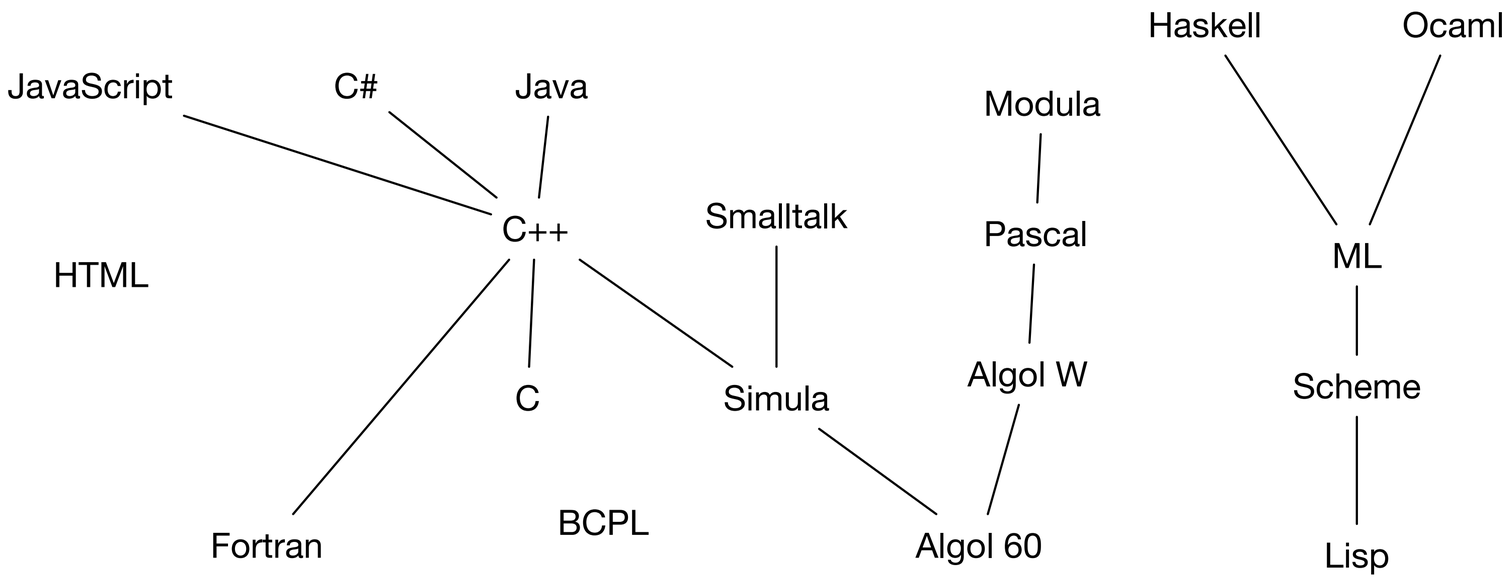
- `assert` permet de vérifier des assertions dans un programme et génère une erreur si non vérifiée

Python - recap !

- beaucoup d'exemples en python avec Google ou autre indexeur
- lire les tutoriels, par exemple `http://www.w3schools.com/python/`
- utiliser `help()` en mode terminal, et taper `q` pour sortir du mode help
- sous `help()` et taper `modules` pour avoir la liste des modules
- faire `import random` pour charger le module
- et `help(random)` donne une info sur ce module
- et `help(random)` donne une info sur ce module
- aussi `dir(random)` donne toutes les méthodes et attributs de la classe `random`

Les langages de programmation

- généalogie des langagses de programmation



à faire

- analyses lexicale et syntaxique
- modularité et programmation objet
- programmation graphique
- algorithmes géométriques
- calculs flottants et méthodes numériques
- programmation de plusieurs fils de calcul
- assertions et logique des programmes
- introduction à l'informatique théorique
- etc

vive l’informatique

et

la programmation !