INFO2

Magistère de Mathématiques de Rennes
2ème année

26 janvier 2016

école normale supérieure rennes

UNIVERSITÉ DE RENNES 1

Dans l'épisode précédent…

- Deux modes d'exécution
  - compilation
  - interprétation
- Trois types de bases
  - entiers
  - flottants
  - booléens

Flottants : quizz

Qu'est-ce que ce programme va afficher ?

```
x = 0.0
for i in range(10):
    x = x + 0.1
if x == 1.0:
    print x, ' est egal a 1.0'
else:
    print x, ' n\'est pas egal a 1.0'
```

```
1.0  n'est pas egal a 1.0
```

Quelle représentation machine pour 0.1 ?

- Rappel

$$F(x) = (-1)^s \times M \times 2^{e-127}$$

$$M = \begin{cases} 1, f_{23} f_{22} \cdots f_0 & e \neq 0, e \neq 255 \\ 0, f_{23} f_{22} \cdots f_0 & e = 0 \end{cases}$$

- il n'est pas possible de représenter ce nombre de manière exacte !

Explications

- Lors des additions successives, c'est le représentant de 0.1 qui est ajouté
- Le résultat final n'est pas égal à 1.0
- … est n'est pas non plus le représentant flottant de 1.0 !
- L'affichage flottant de Python se permet de faire un arrondi et affiche 1.0 au lieu de 0.999999999999…

Flottants : moralité

- Vigilance sur le cumul d'erreurs d'arrondis
- Ne jamais faire de tests d'égalités sur les flottants
- Ne pas prendre l'affichage des flottants pour « argent comptant »

Entiers/flottants : les conversions de types

- Conversions explicites
  - flottants vers entiers

```
float(2353)
```
→ `2353.0`

  - entiers vers flottants

```
int(3.14)
```
→ `3`

perte de précision !

- Conversions implicites

```
1.2 + 3
```
→ `3.2`

python 2.7

```
1 / 4
```
→ `0`

```
1 / 4.
```
→ `0.25`

```
1 // 4
```
→ `0`

```
1 // 4.
```
→ `0.0`

Conversions de types (numériques) : moralité

- Certains opérateurs utilisent le même symbole mais changent de comportement en fonction du type des arguments

- Le type d'une valeur peut être changé explicitement (si son contenu le permet)

- Dans certains langages, les types peuvent être convertis implicitement : élégant, mais parfois déroutant !

Flot de contrôle

Flot de contrôle

Les boucles

Boucles *for*

compter jusqu'à trois…

```
for i in [1 ,2 , 3]:
    print i
```

en sortie de boucle, i vaut 3

```
for i in range(3):
    print i+1
```

range(n) construit la liste [0, .., n -1]

```
for i in range(1,4):
    print i
```

range(a,n) construit la liste [a, .., n -1]

```
for i in range(2,7,2):
    print i/2
```

range(a,n,k) construit la liste [a, .., n -1], mais ne garde que les multiples de k

Exercice

Écrire une fonction `index(t,e)` qui renvoie la position de l'élément `e` dans le tableau `t`.
(sans utiliser d'instruction `return` dans la boucle principale)

Exercice

Écrire une fonction `index(t,e)` qui renvoie la (première) position de l'élément `e` dans le tableau `t`, ou `-1` si l'élément `e` n'est pas présent dans `t`.
(sans utiliser d'instruction `return` dans la boucle principale)

Solution #0

```
def find(e,t):
    for i in range(len(t)):
        if t[i]==e:
            return i
return -1
```

l'instruction **`return`** interrompt la fonction, et par la même occasion, la boucle

mais on veut parfois seulement interrompre la boucle et pas toute la fonction…

Solution #1

```
def find(e,t):
    for i in range(len(t)):
        if t[i]==e:
            break
    if t[i]==e:
        return i
    else:
        return -1
```

l'instruction **break** interrompt la boucle

il faut comprendre par quelle porte nous sommes sortis de la boucle

Solution #2

```
def find(e,t):
    i = 0
while (i<len(t) and t[i]!=e):
    i += 1
if i < len(t):
    return i
else:
    return -1
```

La boucle `while` généralise la boucle `for`

Attention aux débordements de tableaux !

l'opérateur `and` est paresseux : le deuxième argument n'est pas évalué si le premier renvoie `False`

Exercice

Écrire une fonction `enter_str_gt4()` qui demande à l'utilisateur une entrée interactive (avec la fonction `raw_input(msg)`) et renvoie le résultat si la chaîne correspondante possède au moins 4 caractères. La fonction ne termine pas tant que l'utilisateur n'a pas saisie une chaîne adaptée.

Solution

```
def enter_str_gt4():
    while True:
        v = raw_input('Entrer un texte d\'au moins 4 caractères : ')
        if len(v) >= 4:
            break
        print 'texte trop court, essaye encore !'
    return v

res = enter_str_gt4()
print 'résultat =', res
```

boucle infinie

unique sortie de la boucle

Moralité

- Privilégier si possible les boucles `for`, mais éviter cependant de trop vous appuyer sur la valeur finale de la variable d'itération (peut varier en fonction des langages)

- En l'absence de construction `repeat/until`, utiliser une boucle `while` infinie + `break`.

Flot de contrôle

Les fonctions

Bonnes pratiques

- Un programme trop gros devient difficile à comprendre et à débuguer

- Il faut le découper en petites **fonctions** élémentaires

  - chaque fonction aura une *spécification* claire

  - chaque fonction sera testée individuellement (*test unitaire)*

Arguments

passage par position

```
def print_name(first,last,reverse):
    if reverse:
        print last, first
    else:
        print first, last

print_name('David','Pichardie',False)
print_name('David','Pichardie',True)
```

Arguments

passage par nom

```
def print_name(first,last,reverse):
    if reverse:
        print last, first
    else:
        print first, last

print_name(reverse=False,
           first='David',
           last='Pichardie')
```

Arguments

valeurs par défaut

```
def print_name(first,last,reverse=False):
    if reverse:
        print last, first
    else:
        print first, last

print_name('David','Pichardie')
```

Ordre d'évaluation

```
def print_sum(i,j):
    print (i+j)
    return i+j

def print2():
    print 2
    return 2

def print1():
    print 1
    return 1

v = print_sum(print1(),print2())

print 'v =', v
```

1re évaluation

2e évaluation

3e évaluation

Portée

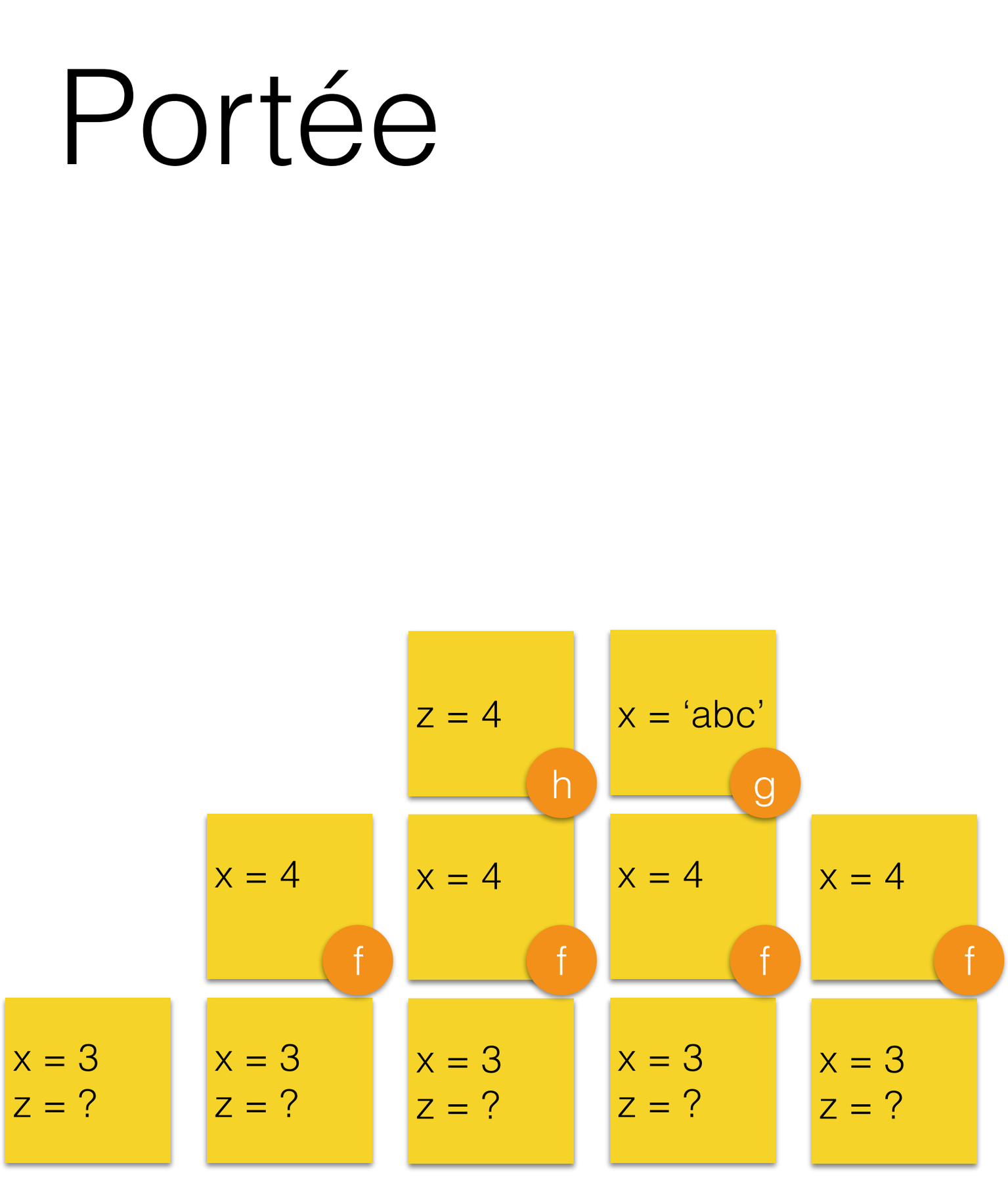
```
def f(u):
    y = 1
    u = u + y
    print 'u =', u
    return u

x = 3
y = 2
z = f(x)
print 'z =', z
print 'x =', x
print 'y =', y
```

Portée

```
def f(x):
    def g():
        x = 'abc'
        print 'x =', x
    def h():
        z = x
        print 'z=', z
        x = x + 1
        print 'x =', x
        h()
        g()
        print 'x =', x
        return g

x = 3
z = f(x)
print 'x =', x
print 'z =', z
z()
```



Portée : moralité

- Il généralement suffisant de manipuler les variables locales de la fonction courante

- Éviter aussi les globales (attention à la syntaxe Python)

Variables globales

sans l'annotation `global`, la variable `count` serait locale !

```
def fib(n):
    global count
    count = count +1
    if n ==0 or n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

count = 0
print "fib(10) =", fib(10)
print "#appel fib =", count
```