

### III - Conditionnelles

- expressions booléennes
- opérateurs de comparaison
  - égalité
  - différence
  - opérateurs de comparaison
- opérateurs booléens
  - négation
  - conjonction
  - disjonction
  - prédicat de type
- définition d'un prédicat
- expressions conditionnelles : si, cas
- fonction d'entrée/sortie

### Conditionnelles

- langages permettent calcul de booléens
- booléens indispensables pour conditionnelles :
- exécution que si une condition est vérifiée (i.e  $b > 5$ )

### Conditionnelles

- exemple :
- #Spécification :  $f : \text{réel} \rightarrow \text{réel}$ 
  - #  $x \rightarrow 1 / (x - 2)$
- def  $f(x)$  :
  - return  $1 / (x - 2)$
- problème si  $x = 2$ , division par zéro
- conditionnelle pour éviter ce cas

### Expressions booléennes

- expression de type booléen (= vrai ou faux)
- plus simples : True et False
- autres construites à base de :
  - opérateurs de comparaison
  - opérateurs booléens ou logiques
  - prédicats
- expressions booléennes permettent constructions fonctions booléennes, prédicats et expressions conditionnelles

### Opérateurs de comparaison

- égalité
- en python :
  - $a == b \rightarrow$  rend un booléen
- opérateur à deux opérandes de type quelconque, rend vrai ou faux
  - $1 == 2$
  - $1 == 1$
  - $1 == 2/2$
  - $1 == "1"$
  - $1 == "a"$

### Différence

- différence : opposée de l'égalité
- $a != b \rightarrow$  rend un booléen
- $1 != 2$
- $1 != 1$
- $1 != 2/2$
- $1 != "1"$
- $1 != "a"$

## Opérateurs de comparaison

- égalité et différence arguments de type quelconque
- opérateurs de comparaison aussi mais il faut relation d'ordre
- $x < y$
- $x \leq y$
- $x > y$
- $x \geq y$
- rendent un booléen

## Opérateurs de comparaison

- $1 \leq 2$
- $1 \geq 1.$
- $\text{True} > \text{False}$
- "accord" < "accueil" (ordre lexicographique)

## Opérateurs booléens

- opérateurs dont les opérandes et le résultat sont des booléens
- non : booléen  $\rightarrow$  booléen
- opérateur unaire (en Python : not)
- $\text{not } 1 == 2$  (équivalent à  $\text{not } (1 == 2)$  et pas à  $(\text{not } 1) == 2$ )
- $\text{not } 3 \geq 2 ** 2$  (équivalent à  $3 < 2 ** 2$ )

## Opérateurs booléens

- et : booléen x booléen  $\rightarrow$  booléen
- table de vérité de la conjonction
- | x    | y    | x et y |
|------|------|--------|
| vrai | vrai | vrai   |
| vrai | faux | faux   |
| faux | vrai | faux   |
| faux | faux | faux   |
- en Python opérateur binaire and
- $2 \leq 3$  and  $3 \leq 7$
- $\text{not "aa"} > \text{"a"} \text{ and } 10\%5 \neq 0$
- $\text{not ("aa"} > \text{"a"} \text{ and } 10\%5 \neq 0)$

## Opérateurs booléens

- ou : booléen x booléen  $\rightarrow$  booléen
- table de vérité de la disjonction
- | x    | y    | x ou y |
|------|------|--------|
| vrai | vrai | vrai   |
| vrai | faux | vrai   |
| faux | vrai | vrai   |
| faux | faux | faux   |
- en Python opérateur binaire or
- priorité : not, and puis or
- $a\%2 == 0$  and  $a\%3 == 0$  or  $a\%5 == 0$
- $a\%2 == 0$  and ( $a\%3 == 0$  or  $a\%5 == 0$ )

## Prédicat de type

- isinstance : indifférent x type  $\rightarrow$  booléen
- permet de tester si le premier paramètre est du type donné en second paramètre
- valeurs possibles pour le type :
- bool
- int
- float
- str
- isinstance ( $2**64 - 2**64$ , int)
- isinstance ( $2**64 \geq 0$ , bool)
- isinstance ( $2**64$  and 0, bool)

## Définition d'un prédicat

```
fonction dont le résultat est booléen (nom terminé par Q) :
## interieurQ : nombre x nombre x nombre (>0) -> booléen
#           x, y, r -> (x,y) est dans (0, 0, r)
import math
def interieurQ(x,y,r):
    #Indique si le pt (x,y) est à l'intérieur du cercle de centre (0,0) et
    #de rayon r
    return math.sqrt(x**2+y**2)<r

#programme principal : test de la fonction interieurQ
print ("Le point (.5,.5) est dans le cercle de rayon 1 :", interieurQ
(.5,.5,1))
```

Ifsisc – Univ Rennes

60

## Expressions conditionnelles

- "si" : expression conditionnelle, résultat = évaluation expression(s) booléenne(s) : si, cas
- "si" en Python "if" avec ou sans "else" (sinon)
- ex : nombre supérieur à 100 ?

```
val = 99
if (val > 100) :
    print ("la valeur dépasse 100")
    # avec un sinon :
    val = 99
if (val > 100) :
    print ("la valeur dépasse 100")
else :
    print ("la valeur ne dépasse pas 100")
```

Ifsisc – Univ Rennes

61

## Expressions conditionnelles

- #monMax : nombre x nombre -> nombre
- # a, b ->
- def monMax(a,b):
- #rend le nombre le plus grand entre a et b
- if (a>b): return a
- else: return b
- # programme principal
- print ("Le maximum de 2 et 3 est", monMax(2,3))
- NB : fonction max existe déjà en Python
- max : nombre x ... x nombre -> nombre

Ifsisc – Univ Rennes

62

## Expressions conditionnelles

- imbrications possibles
- attention à l'indentation !!!

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

Ifsisc – Univ Rennes

63

## Expressions conditionnelles

- if (val==4):
- print ("val = 4")
- if (v==2):
- print("test sur v")
- else:
- print("test sur v diff de 2")
- else:
- print("val != 4")

Ifsisc – Univ Rennes

64

## Expressions conditionnelles

- cas : en Python mise en œuvre avec if elif :

```
#date: entier -> chaîne de caractères
# an -> évènement marquant date
def date (an):
    "Rend l'évènement marquant de l'année an"
    if (an == 1515):
        return "Bataille de Marignan"
    elif (an == 800):
        return "Sacre de Charlemagne"
    elif (an == 2000):
        return "Fin du XXe siècle"
    else:
        return "Cela dépasse mes compétences"
```

Ifsisc – Univ Rennes

65

## Expressions conditionnelles

- test :
- `print (date (1515))`
- Bataille de Marignan
- `print (date (2.5))`
- Cela dépasse mes compétences
- on termine le cas, non pas avec `elif` mais avec `else` de manière à couvrir tous les cas

## Fonctions d'entrée / sortie

- compléments sur la fonction `print` :
- si dans la liste des paramètres, on ajoute « `end=""` » l'affichage suivant sera sur la même ligne
- « `\n` » permet de passer à la ligne
- `input()` attente de saisie d'une chaîne de caractères terminée par « entrée »
- si une chaîne de caractères en paramètre de `input()`, elle sera affichée avant la saisie
- pour obtenir un nombre il faut utiliser `int()` ou `float()` pour transformer la chaîne en nombre

## Fonctions d'entrée / sortie

```
res = input("Une chaîne SVP : ")
print (res)
```

```
n = int(input("Un entier SVP :"))
print(n**2)
```

```
reel = float(input("Un nombre SVP :"))
print(reel*2)
```

```
input("« appuyer sur une touche pour continuer »")
print("Un message sans retour à la ligne : ", end="")
print(" la preuve !")
```