Informatique 1
Introduction à la programmation

Mission 1 : RESTRUCTURATION

Programmes, Variables & Valeurs, Expressions & Instructions

Kim Mens Siegfried Nijssen Charles Pecheur

Mission 1 : restructuration

- Programmes, instructions & expressions
- Variables
- Instructions conditionnelles : if, else, elif
- Instructions de boucle : while
- Entiers, réels, booléens

Programme & Instructions

```
1 capital = 1000
2 taux = 5
3 duree = 20
4 an = 0
5 while an <= duree :
      print(an, "\t", capital)
      capital = capital * (1 + taux/100)
      an = an+1
```

Un programme = une **séquence d'instructions**

Programme & Instructions

```
capital = 1000
taux = 5
duree = 20
an = 0
while an <= duree :
    print(an, "\t", capital)
    capital = capital * (1 + taux/100)
    an = an+1
```

Un programme = une **séquence d'instructions**

Exécuter un programme =

exécuter les instructions l'une après l'autre

Programme & Instructions

```
capital = 1000
taux = 5
duree = 20
an = 0
while an <= duree :
    print(an, "\t", capital)
    capital = capital * (1 + taux/100)
    an = an+1
```

Un programme = une **séquence d'instructions**

Exécuter un programme =

exécuter les instructions l'une après l'autre

Exécuter une instruction produit un **effet**

Expressions

Evaluer une expression calcule **un résultat**

```
capital * (1 + taux/100)
an + 1
"Python 3"
max(x, y, z)
n
```

Expressions **simples** (atomiques) :

constante (litéral), variable

Expressions **composées** :

opérations, fonctions

Conditions

Une **condition** = une **expression**

dont le résultat calculé est un **booléen**:

une valeur logique VRAI (**`True`**) ou FAUX (**`False`**)

```
x != 0
answer == "yes"
(x - x_0) ** 2 < epsilon
not (x < 10 or y > 0)
```

Typiquement utilisé dans les instructions de contrôle :

```
if condition: … else: …
while condition: …
```

Expression ≠ Instruction

Exécuter une instruction produit un **effet**

```
duree = 20
print(an, "\t", capital)
```

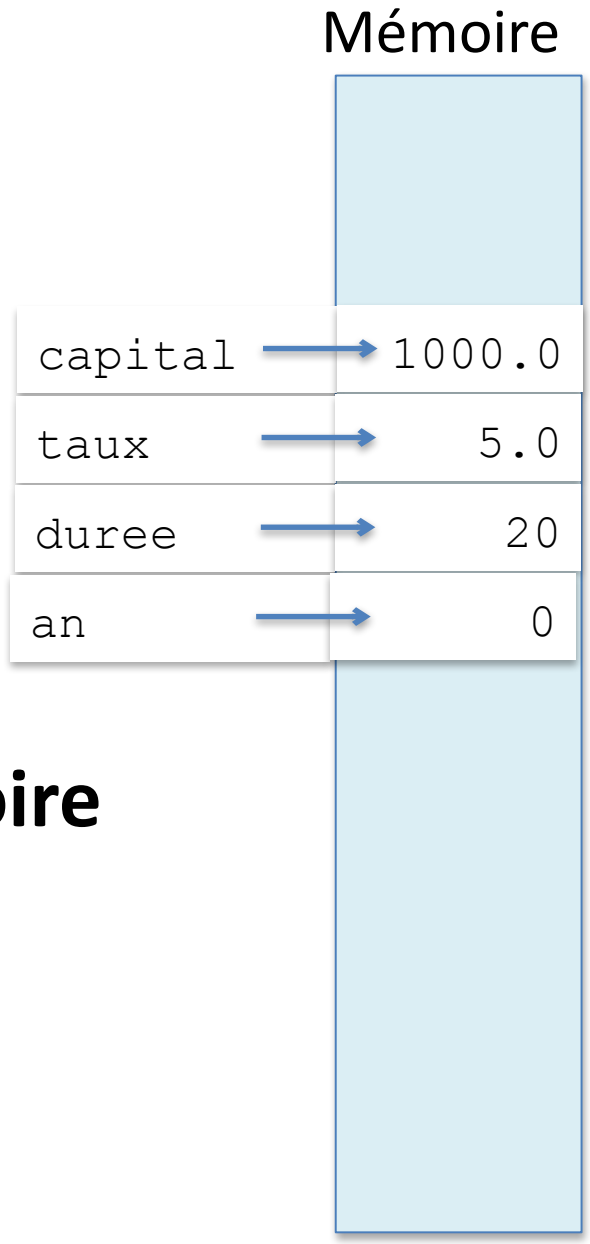
Evaluer une expression calcule **un résultat**

```
capital * (1 + taux/100)
an + 1
```

sans affectation, print, … ce résultat est immédiatement perdu!

Variables

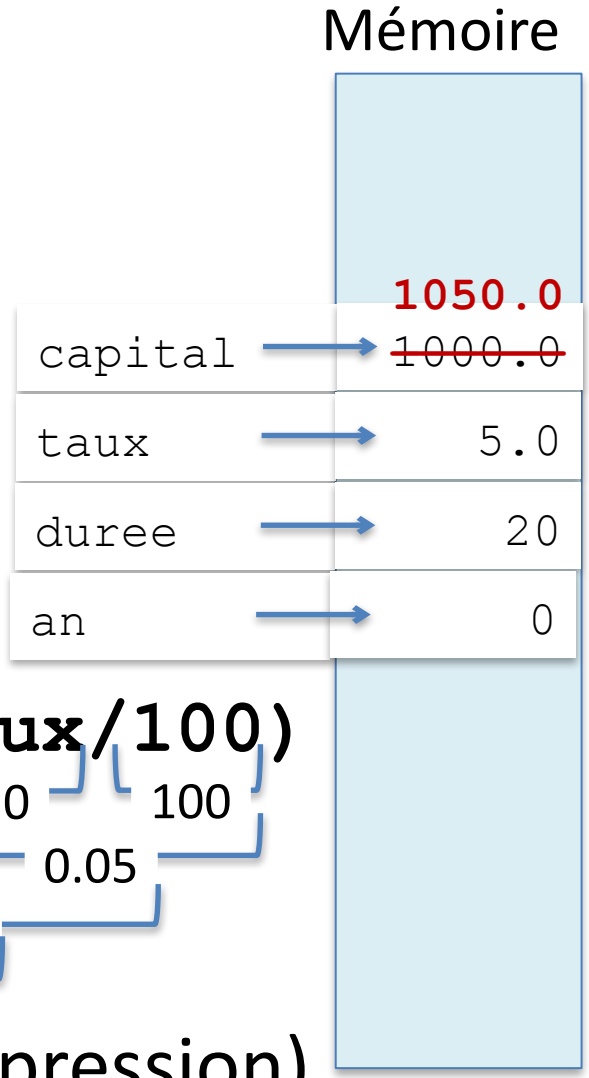
```
capital = 1000.0
taux = 5.0
duree = 20
an = 0
while an <= duree :
    print(an, "\t", capital)
    capital = capital * (1 + taux/100)
    an = an+1
```



Une **variable** = une zone de **mémoire**

Affectation

```
capital = 1000.0
taux = 5.0
duree = 20
an = 0
while an <= duree :
    print(an, "\t", capital)
    capital = capital * (1 + taux/100)
    an = an+1
```

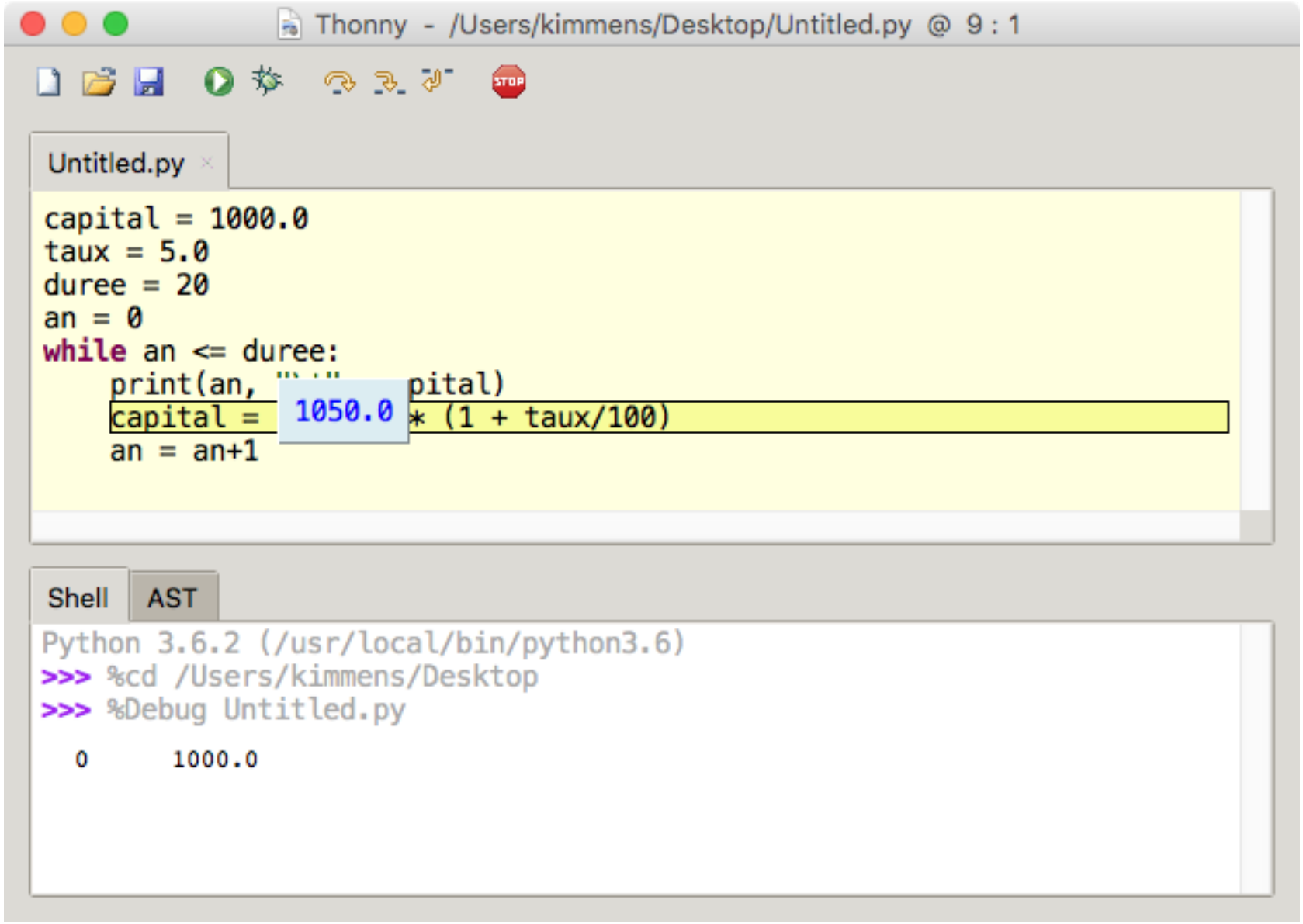


capital = capital * (1 + taux/100)

1000.0 — 1 — 5.0 — 100 — 0.05 — 1.05 — 1050.0

Lire la variable : **utilisation** (une expression)

Modifier la variable : **affectation** (une instruction)


Thonny - /Users/kimmens/Desktop/Untitled.py @ 9 : 1
Untitled.py
capital = 1000.0
taux = 5.0
duree = 20
an = 0
while an <= duree:
print(an,
pital)
1050.0
capital =
* (1 + taux/100)
an = an+1
Shell
AST
Python 3.6.2 (/usr/local/bin/python3.6)
>>> %cd /Users/kimmens/Desktop
>>> %Debug Untitled.py
0 1000.0

Instructions de contrôle

```
capital = 1000
taux = 5
duree = 20
an = 0
while an <= duree :
    print(an, "\t", capital)
    capital = capital * (1 + taux/100)
    an = an+1
```

Contrôlent l'exécution du programme

Instructions de contrôle

```
capital = 1000
taux = 5
duree = 20
an = 0
while an <= duree :
    print(an, "\t", capital)
    capital = capital * (1 + taux/100)
    an = an+1
```

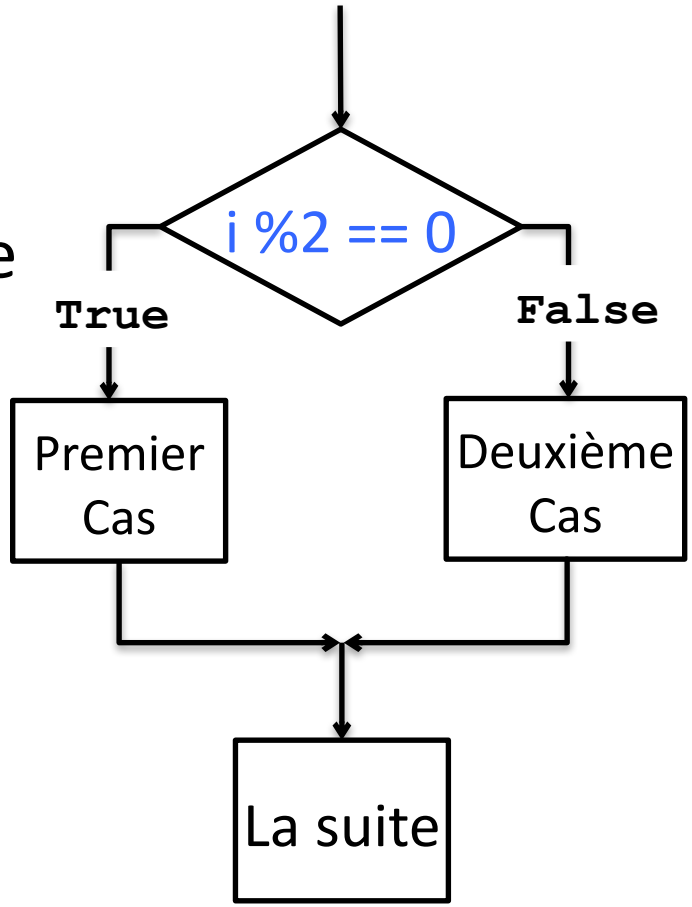
Contrôlent l'exécution du programme

Instructions composées

Instruction conditionnelle (if-else)

```
i = 0
# Deux cas :
# i pair, i impair
if i % 2 == 0 :
    print(i,"est pair")
else :
    print(i,"est impair")
#
# La suite
```

(1) Cas
(3) if
(5) else



Instruction conditionnelle (if-else)

```
if x > y :
    x = x-1
    y = y-1
else :
    x = x-1
    y = y+1
```

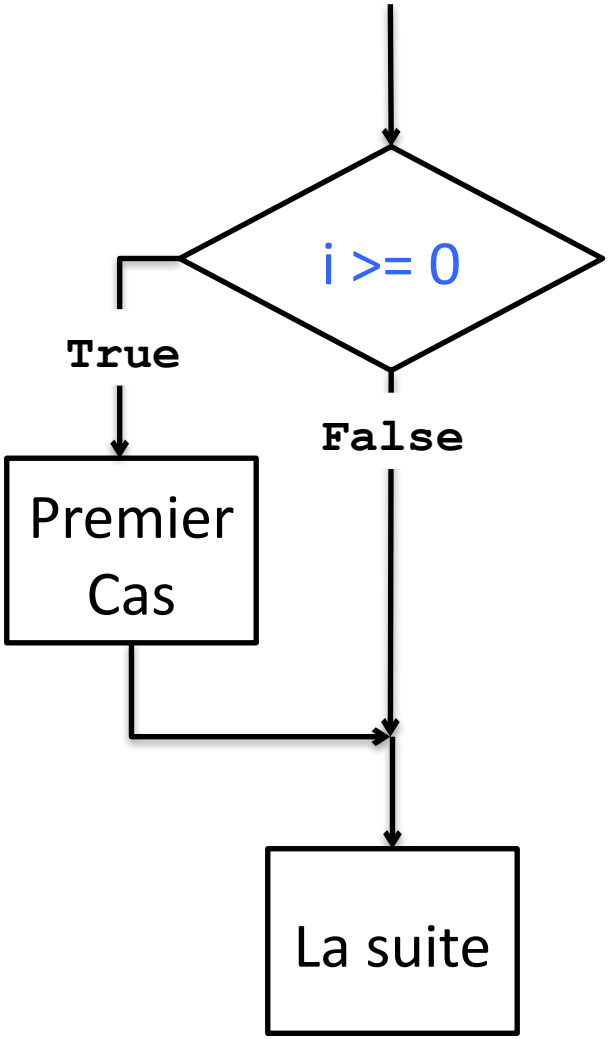
```
if x > y :

    y = y-1
else :

    y = y+1
x = x-1
```

Instruction conditionnelle (if)

```
i = 0
# Un seul cas :
# i >= 0
if i >= 0 :
    print(i,"est pair")
#
# La suite
```

(1) Cas

(3) if



Instruction conditionnelle (if)

```
if x > y :
    x = x-1
    y = y-1
else :
    x = x-1
```

```
if x > y :
    y = y-1

x = x-1
```

Conditionnelle en chaîne

```
if x > y :
    x = x-1
else :
    if x < y :
        x = x+1
    else :
        print(x)
```

```
if x > y :
    x = x-1
elif x < y :
    x = x+1
else :
    print(x)
```

Instruction conditionnelle (if-elif-else)

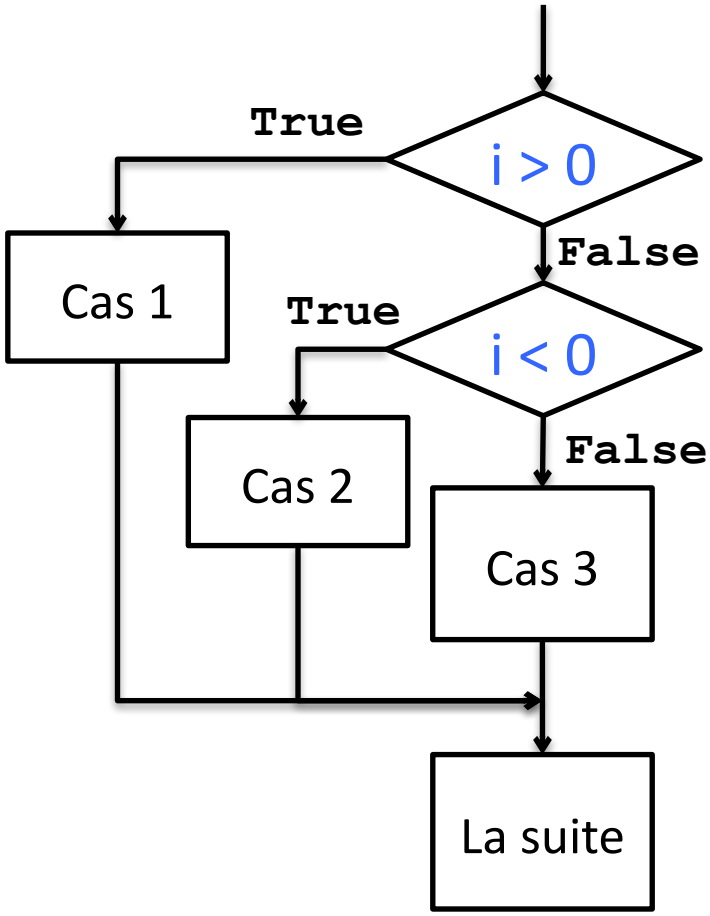
```
i = 0
# Trois cas : i<0
# i==0 et i > 0
if i > 0 :
        print("i est positif")
elif i < 0 :
        print("i est négatif")
else :
        print("i est null")
#
# La suite
#
```

(1) Cas

(3) if

(4) elif

(5) else



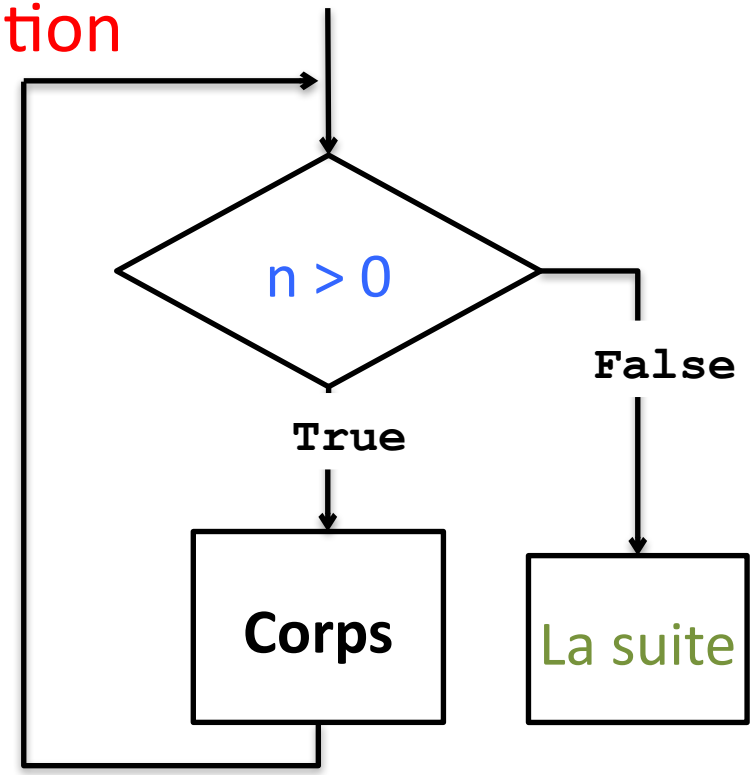
Instruction de boucle

```
# Quel est la somme des 10 premiers carrés ?
sum = 0
n = 10
while n > 0 :
    sum = sum + n*n
    n = n - 1
#
# La suite
#
```

(1) Initialiser

(2) Condition

(3) Corps



Instruction de boucle (while)

```
i = 0
n = 5
while i < n:
   print ( "Bonjour" )
   i = i + 1
```

```
Bonjour   i = 0
Bonjour   i = 1
Bonjour   i = 2
Bonjour   i = 3
Bonjour   i = 4
          i = 5
```

Instruction de boucle (while)

```
i = 1
n = 5
while i <= n:
   print ( "Bonjour" )
   i = i + 1
```

```
Bonjour   i = 1
Bonjour   i = 2
Bonjour   i = 3
Bonjour   i = 4
Bonjour   i = 5
          i = 6
```

Instruction de boucle (while)

```
i = 0
n = 5
while i <= n:
   print ( "Bonjour" )
   i = i + 1
```

```
Bonjour   i = 0
Bonjour   i = 1
Bonjour   i = 2
Bonjour   i = 3
Bonjour   i = 4
Bonjour   i = 5
          i = 6
```

Instruction de boucle (while)

```
i = 1
n = 1
while i < 10:
    i = i + 1
    n = n + 1
    print(i)
```

```
i = 1

while i < 10:
    i = i + 1

    print(i)
```

Blocs

```
i = 1
sum = 0
while sum < 10 :
    sum = sum + i
      i = i+1
```

Erreur d'indentation

```
i = 1
sum = 0
while sum < 10 :
    sum = sum + i
i = i+1
```

Correct !
(mais un peu long)

```
i = 1
sum = 0
while i < 10 :
    sum = sum + i
i = i+1
```

Boucle infini

Entiers et réels

Nombres entiers

taille illimitée

```
0, 1, 42, -10, ...
x = 2
3 * x ** 2 + 4 * x - 2
(42 // 5) * 5 + 42 % 5
```

Division entière

Reste de la division entière

Nombres réels

taille et précision limitées
virgule flottante

1,602	-23

```
0.0, 1.0, 0.01, 3e8, 1,602e-23, …
x = 0.5
3.0 * x ** 2 + 4.0 * x - 2.0
(42 / 3) * 3
```

Calculs numériques

```
>>> 10 ** 20
100000000000000000000
>>> 10 ** 20 - 1
99999999999999999999
>>> 10.0 ** 20
1e+20
>>> 10.0 ** 20 - 1
1e+20
>>> 1 - 1/3 - 1/3 - 1/3
1.1102230246251565e-16
>>> 10/0
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ZeroDivisionError: division by zero
```

La représentation des nombres réels en Python est imprécise

https://docs.python.org/3/tutorial/floatingpoint.html

Conditions numériques

Qu'affiche cette boucle ?

```
iter = 0
d = 1
while d != 0.0:
    iter = iter + 1
    d = d - 1/3
# d == 0.0
print(iter)
```

Rien n'est affiché !

boucle infinie !

Conditions numériques

Qu'affiche cette boucle ?

```
iter = 0
d = 1
while abs(d) > 0.0000001:
    iter = iter + 1
    d = d - 1/3
# d =~ 0.0
print(iter)
```

Affiche : 3

Opérations booléennes

Négation logique (**not**)

a	not a
True	**not** True == False
False	**not** False == True

ET logique (**and**)

a / b	True	False
True	True **and** True == True	True **and** False == False
False	False **and** True == False	False **and** False == False

OU logique (**or**)

a / b	True	False
True	True **or** True == True	True **or** False == True
False	False **or** True == True	False **or** False == False