Labo d'introduction à l'informatique

pour les mathématiques

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

Semaine 4.2
Boucles while

Un problème

- Une boucle `for` est idéale quand on connaît le nombre de répétitions que l'on veut effectuer.

- La boucle `for` est souvent le meilleur choix pour itérer sur des tableaux, calculer des sommes finies, etc.

- Mais quand on ne sait pas à l'avance quand la boucle se termine ?
  Exemple : jeu-vidéo, processus aléatoire, dépendance d'un input …

La boucle while

- La boucle while répète une instruction **tant que**…
- … une certaine **condition** est vérifiée.
- Exemple : “tant que le joueur n’a pas perdu, afficher les images du jeu”.
- On définit une nouvelle structure de contrôle nommée `while`.

Exemple de boucle while

Jeu du “deviner le nombre”:

*Le joueur doit deviner un nombre décidé par le programmeur. **Tant que** le joueur n’a pas deviné, il peut proposer un nombre. Lorsque le joueur a deviné, on affiche un message pour lui dire qu’il a gagné.*

Exemple de boucle while

Jeu du “deviner le nombre”:

```
nombre_a_deviner = 12
x = 0
while x != nombre_a_deviner:
    x = int(input("Devinez le nombre entre 1 et 100 : "))

print("Vous avez gagné !")
```

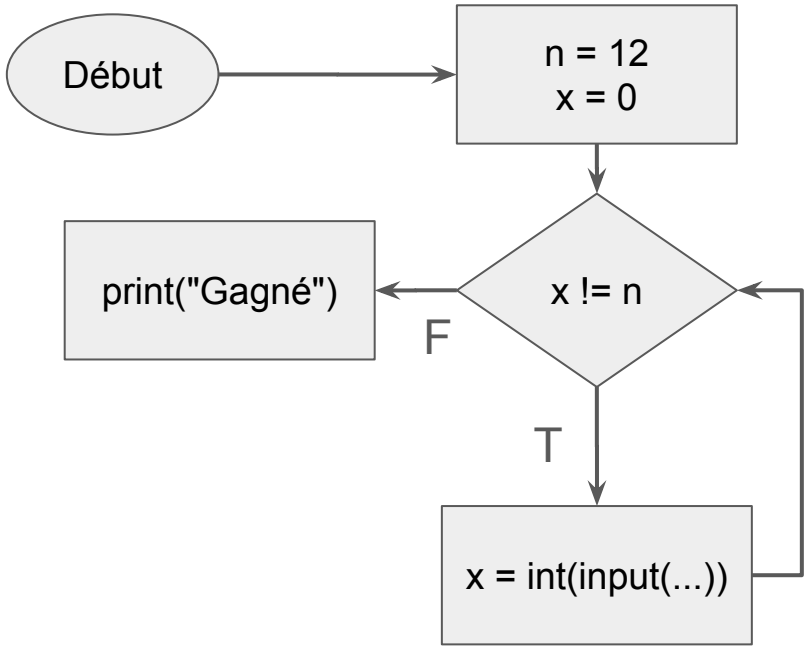
Exemple de boucle while

Jeu du “deviner le nombre”:

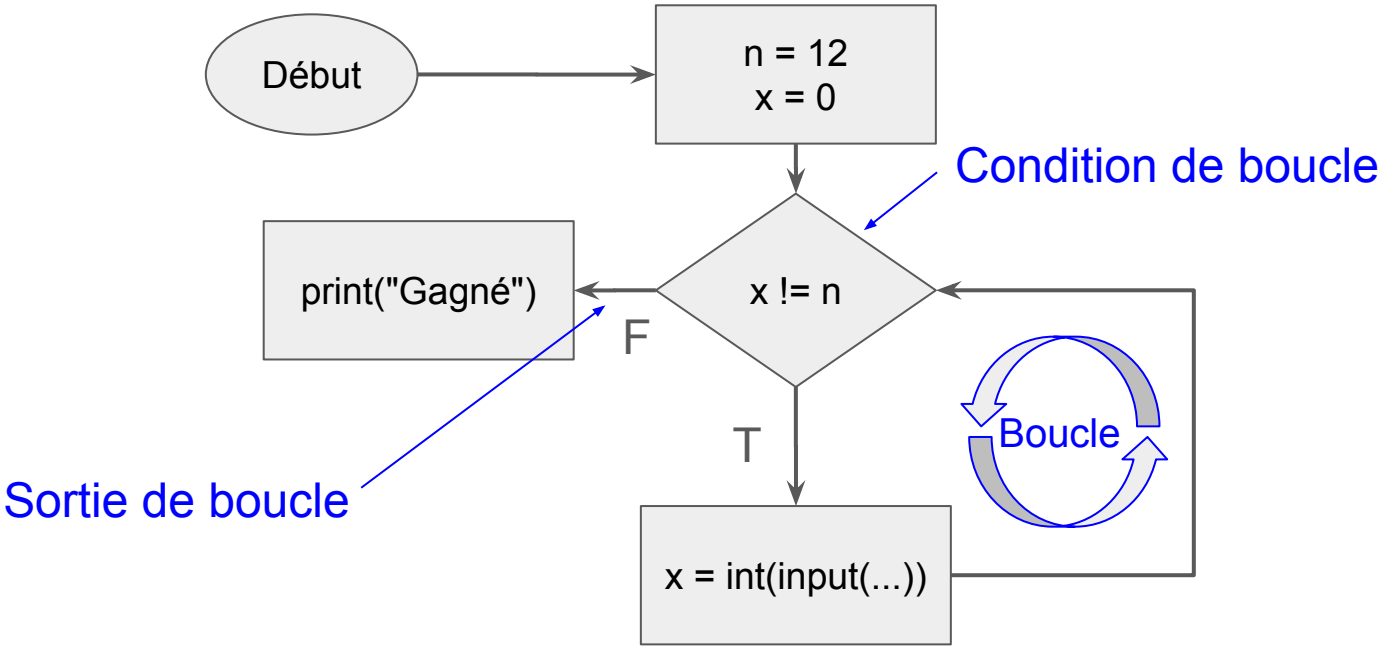
```
nombre_a_deviner = 12
x = 0
while x != nombre_a_deviner:
    x = int(input("Devinez le nombre entre 1 et 100 : "))

print("Vous avez gagné !")
```

On atteint cette ligne uniquement lorsqu’on est libéré du `while`

Logigramme | **Jeu de la devinette**



Logigramme | **Jeu de la devinette**



Sortir d’une boucle while

Jeu du “deviner le nombre”, version 2 :

Le joueur doit deviner un nombre décidé par le programmeur. Tant que le joueur n’a pas deviné, il peut proposer un nombre. Lorsque le joueur a deviné, on affiche un message pour lui dire qu’il a gagné. ***Si le joueur donne un nombre négatif, on arrête la partie (mais le joueur a perdu).***

Sortir d’une boucle while

Comme pour les boucles `for`, on peut utiliser le mot-clé `break` pour sortir de la boucle sans condition.

```
nombre_a_deviner = 12
x = 0
while x != nombre_a_deviner:
    x = int(input("Devinez le nombre entre 1 et 100 : "))
    if x < 0:
        break
print("Vous avez gagné !")


if x > 0:
    print("Vous avez gagné !")
```

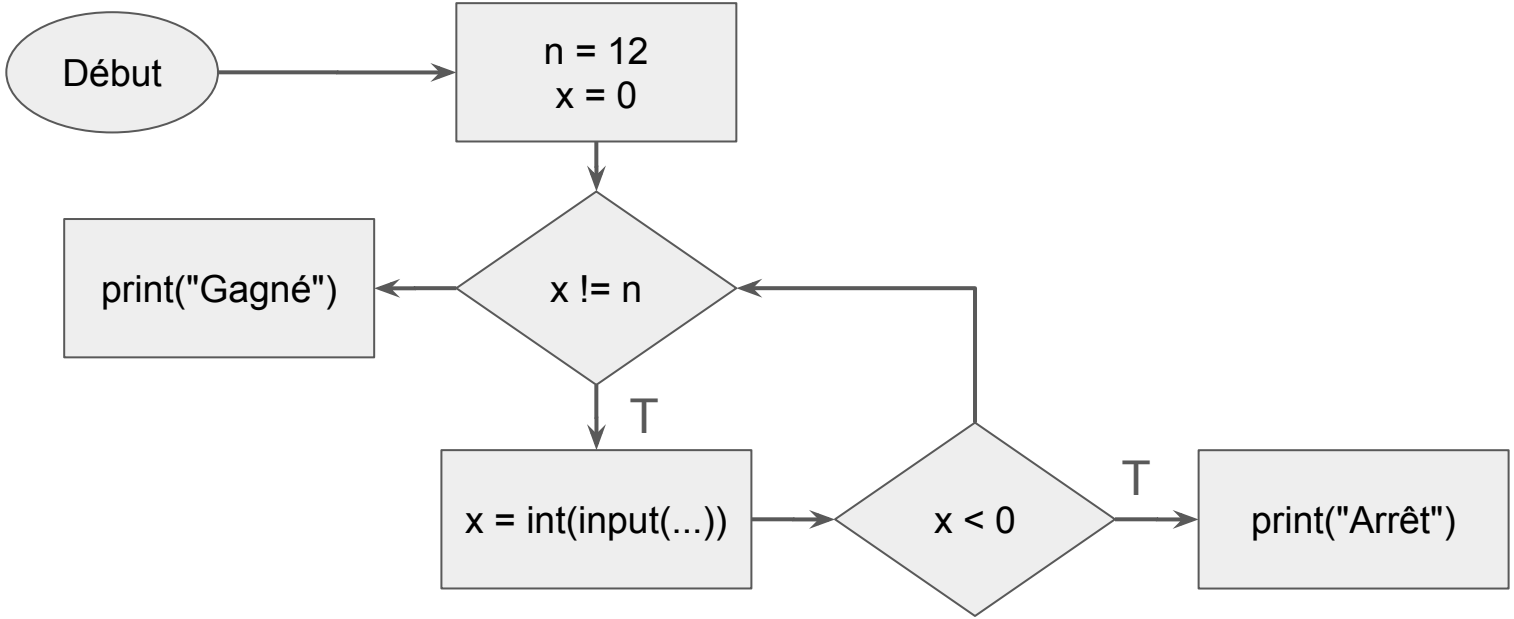
Sortir d’une boucle while

- Comme pour les boucles `for`, on peut utiliser le mot-clé `break` pour sortir de la boucle sans condition.

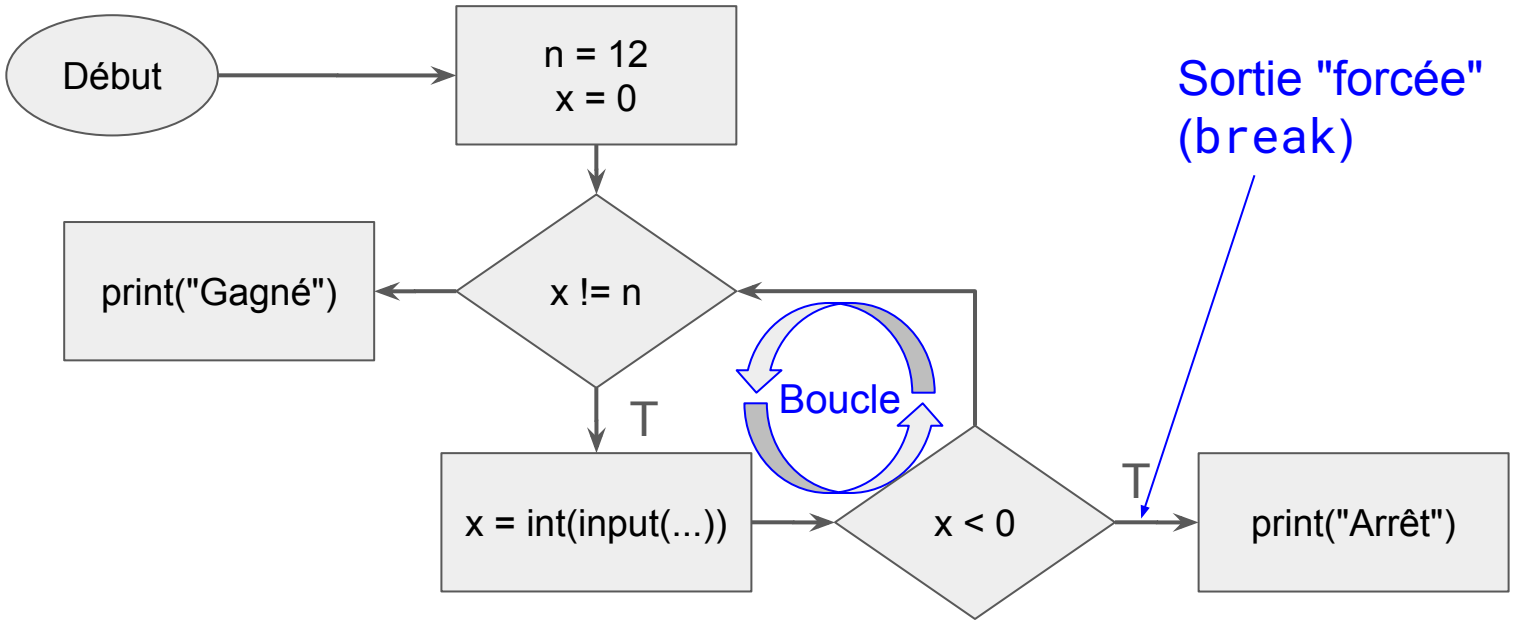
- Cela dit, on peut souvent se passer de `break` en choisissant la condition adéquate :

```
while x >= 0 and x != nombre_a_deviner:
    . . .
```

Logigramme (version avec arrêt prématuré)



Logigramme (version avec arrêt prématuré)



Équivalence entre boucle while et boucle for

On peut toujours écrire un `while` en `for` et vice-versa, mais souvent l'un des deux choix est meilleur que l'autre.

```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

Est équivalent à

```
for i in range(10):
    print(i)
```

Boucles “infinies”

Cas particulier de boucles potentiellement infinies :

```
from random import random
while True: # boucle "infinie" car condition toujours vraie
    x = random() # nombre réel entre 0 et 1
    if x < 0.1:
        break
```

Un autre exemple : méthode de Monte-Carlo

- Il est possible d'estimer la valeur de π en criblant au hasard une cible circulaire avec des "flèches".

- NB : dans certains cas (comme celui que l'on va montrer), il vaut mieux adopter une méthode analytique si celle-ci est envisageable !

Un autre exemple : le marcheur ivre

- Problème du **marcheur ivre** : un homme ivre peut, à chaque étape, avancer d'un pas dans n'importe quel sens (sur un axe donné).

- En moyenne, combien de marcheurs ivres sont-ils repassés au moins une fois par leur point de départ en moins de 10 étapes ?

- Les approches mathématiques livrent des résultats, mais nous nous intéressons ici à une **simulation**.

Un autre exemple : marche aléatoire

Code pour **une** marche aléatoire. Le code suivant contient un danger :

```
from random import random
fini = False
x = 0 # position
pas = 0 # nombre de pas
while not fini:
    pas = pas + 1
    if random() < 0.5:
        x = x + 1
    else:
        x = x - 1
    if x == 0:
        fini  = True
print(pas)
```

Un autre exemple : marche aléatoire

Code pour **une** marche aléatoire. Le code suivant contient un danger :

```
from random import random
fini = False
x = 0 # position
pas = 0 # nombre de pas
while not fini:
    pas = pas + 1
    if random() < 0.5:
        x = x + 1
    else:
        x = x - 1
    if x == 0:
        fini  = True
print(pas)
```

Aucune garantie que cela se termine !

Un autre exemple : marche aléatoire

Code pour **une** marche aléatoire avec nombre maximum d'étapes (ici 10 pour répondre à la question initiale). Dans ce cas, une boucle for ferait également très bien l'affaire.

```
from random import random
fini = False
x = 0 # position
pas = 0 # nombre de pas
max_pas = 10
while not fini and pas < max_pas:
    pas = pas + 1
    if random() < 0.5:
        x = x + 1
    else:
        x = x - 1
    if x == 0:
        fini  = True
if fini:
    print("Terminé en", pas, "pas")
else:
    print("Pas terminé après", pas, "pas, position =", x)
```

Un autre exemple : marche aléatoire

Simulation du taux de marcheurs qui repassent par l'origine en moins de 10 étapes. On encapsule la simulation dans une **fonction** :

```
from random import random
def sim(max_pas):
    x = 0
    pas = 0
    while pas < max_pas:
        pas += 1
        if random() < 0.5:
            x += 1
        else:
            x -= 1
        if x == 0:
            return 1
    return 0

n = 0
for i in range(10_000): #On fait 10'000 simulations
    n += sim(max_pas=10)
print(f"Taux moyen de marcheurs revenant en moins de 10 pas à l'origine: {n/10_000}")
```