Labo d'introduction à l'informatique

pour les mathématiques

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

Semaine 5
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, …

La boucle while

- La boucle while répète une instruction **tant que**…
- … une certaine **condition** est vérifiée.
- Exemple : “tant que le joueur n’a pas quitté, 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 = input(“Devinez le nombre entre 1 et 100 : ”);
end
disp(“Vous avez gagné !”)
```

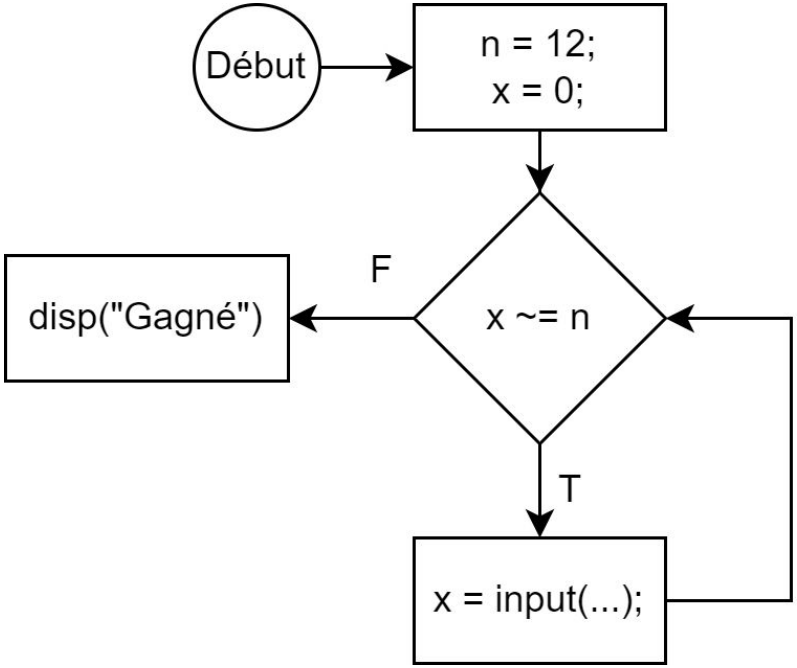
Exemple de boucle while

Jeu du “deviner le nombre”:

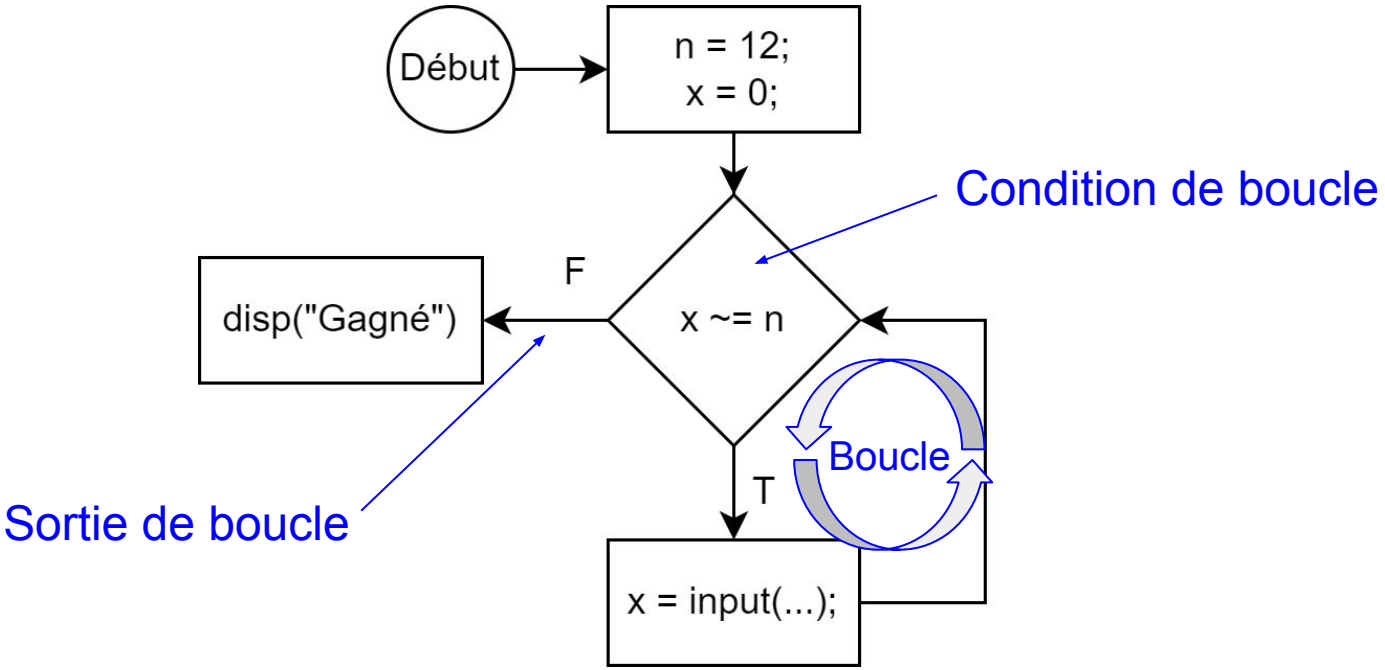
```
nombre_a_deviner = 12;
x = 0;
while x ~= nombre_a_deviner
    x = input(“Devinez le nombre entre 1 et 100 : ”);
end
disp(“Vous avez gagné !”)
```

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 = input(“Devinez le nombre entre 1 et 100 : ”);
    if x < 0
        disp(“Arrêt”) % le joueur veut arrêter de jouer
        break; % interruption de la boucle
    end
end
if x > 0:
    disp(“Vous avez gagné !”)
end
```

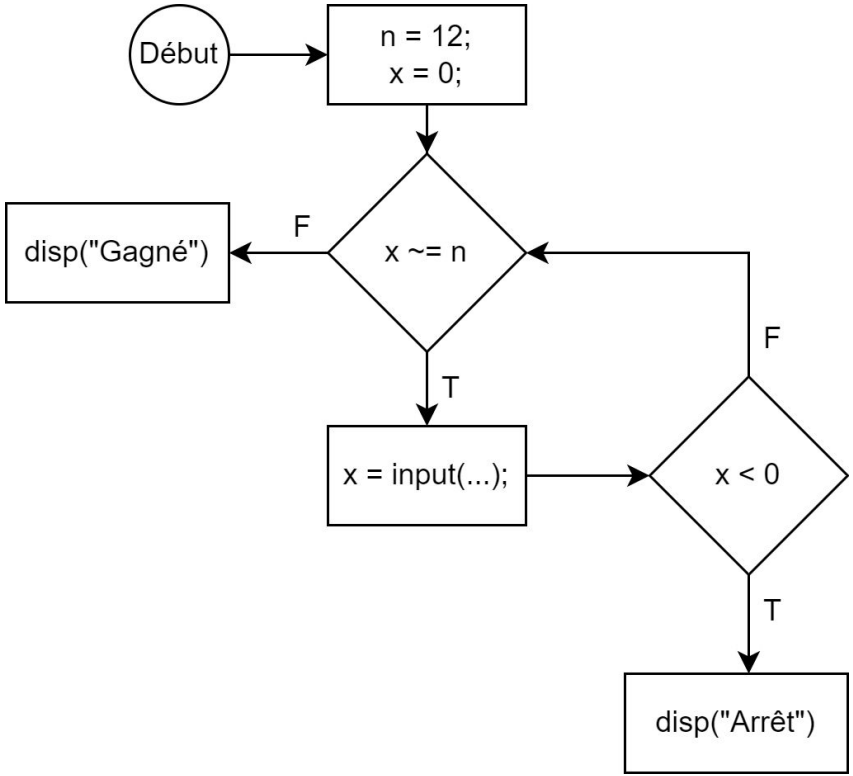
Sortir d’une boucle while

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

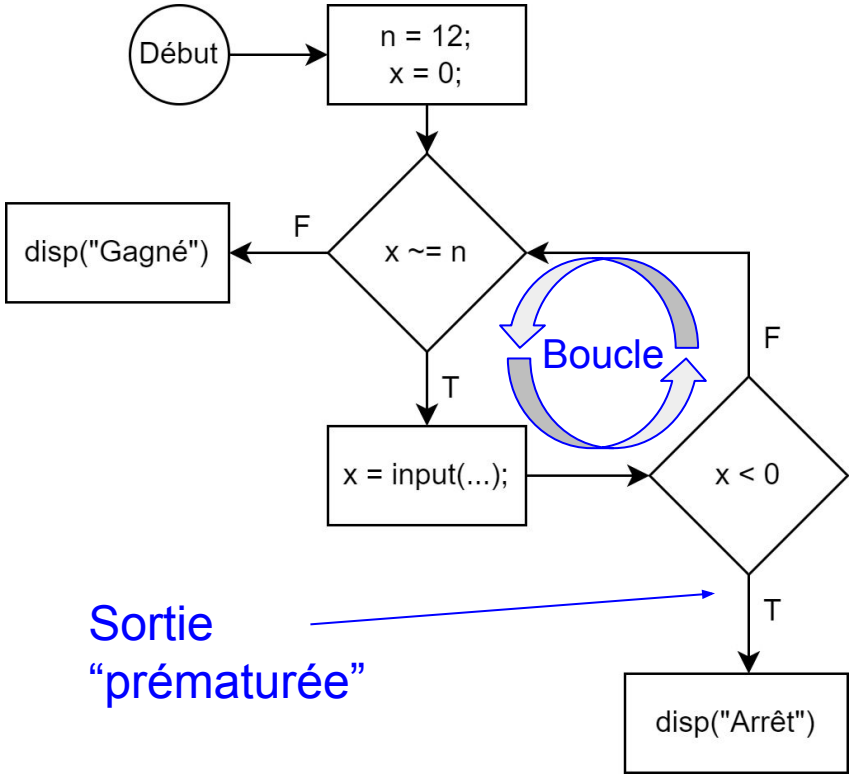
- Cela dit, on peut souvent se passer de `break` en choisissant la condition adéquate :

```
while x >= 0 && x ~= nombre_a_deviner
    . . .
end
```

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 = 1
while i <= 10
    disp(i);
    i = i + 1;
end
```

Est équivalent à

```
for i=1:10
    disp(i);
end
```

Boucles “infinies”

Cas particulier de boucles potentiellement infinies :

```
while true % boucle “infinie”
    x = rand(); % nombre réel entre 0 et 1
    if x < 0.1
        break
    end
end
```

Équivalence entre boucle while et boucle for

S’écrit également (par exemple) :

```
x = Inf;
while x >= 0.1
    x = random();
end
```

ou encore

```
x = Inf;
for i=1:Inf
    x = random();
    if x < 0.1
        break;
    end
end
```

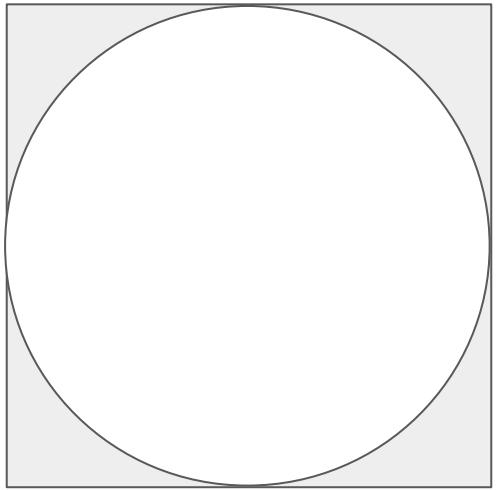
Un autre exemple : méthode de Monte-Carlo

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

- NB : la méthode que l'on va voir n'est **PAS une méthode efficace** pour approximer π, mais elle constitue un exemple simple d'utilisation de nombres pseudo-aléatoires couplés avec des boucles.

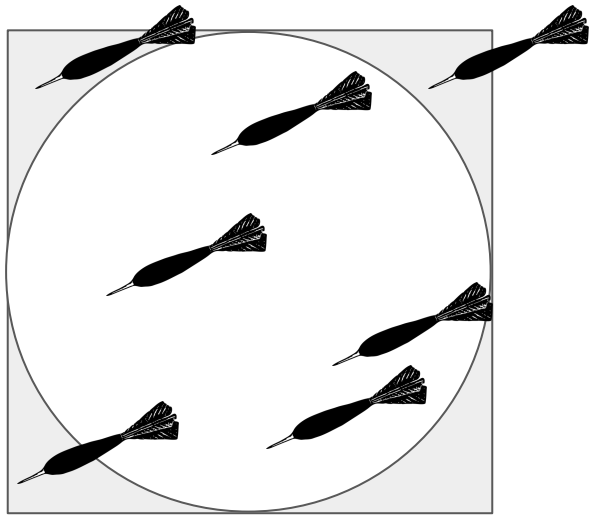
Estimation de π avec un tirage au sort

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



Estimation de π avec un tirage au sort

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



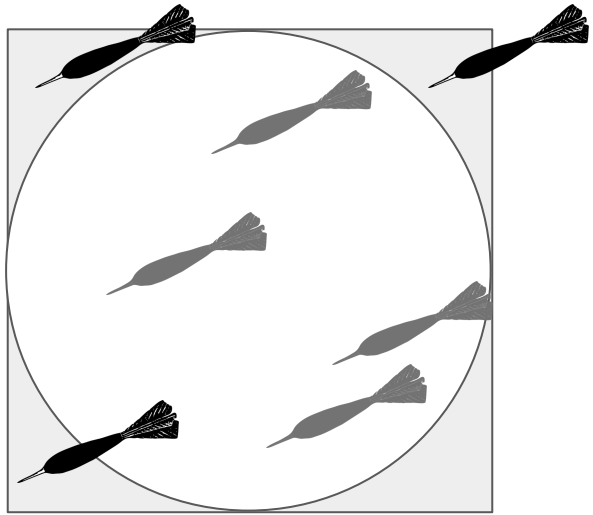
Estimation de π avec un tirage au sort

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

Exemple :

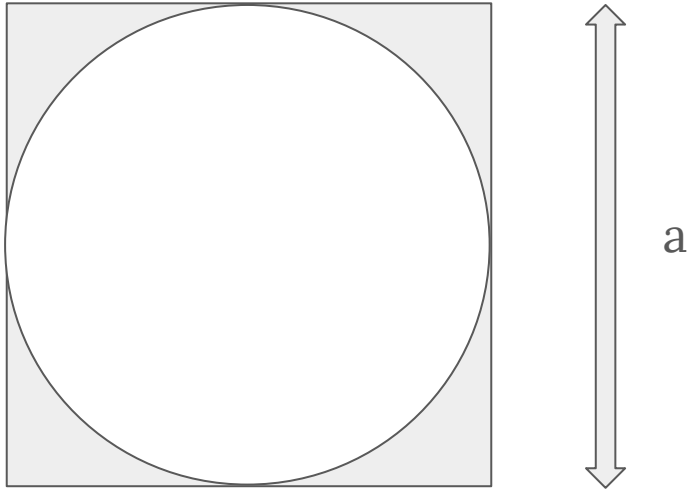
$n_{cible} = 4$

$n_{tot} = 7$



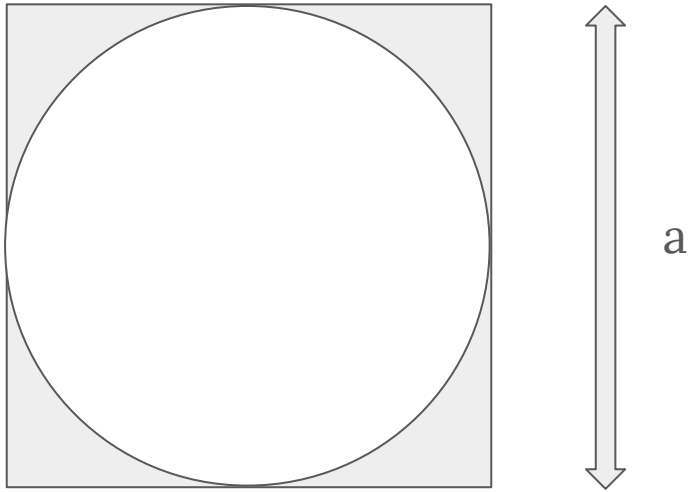
Estimation de π avec un tirage au sort

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



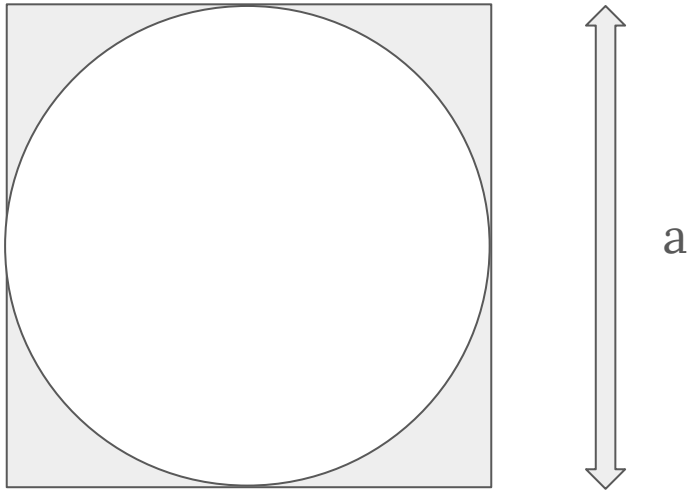
Estimation de π avec un tirage au sort

- $A_s = a^2$
- $A_c = \pi r^2 = \pi a^2/4$



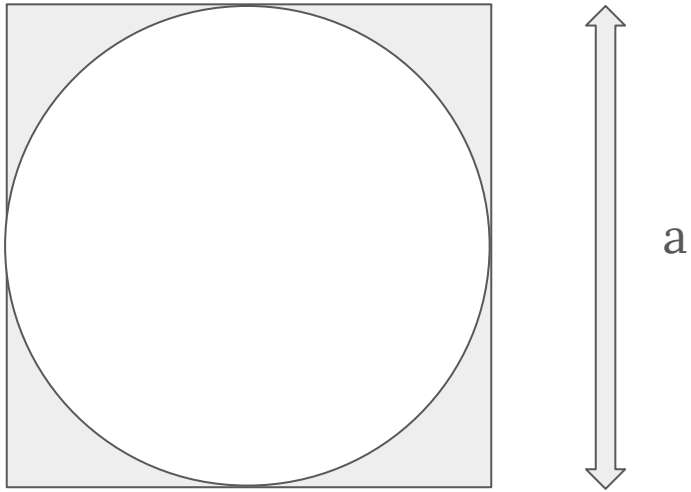
Estimation de π avec un tirage au sort

- Si le nombre de flèches est grand et qu'elles sont réellement lancées aléatoirement, alors la densité de flèche est constante et $A_c \,/\, A_s \approx n_{cible} / n_{tot}$



Estimation de π avec un tirage au sort

$$n_{cible}/n_{tot} \approx A_c / A_s = (\pi a^2/4) / a^2 = \pi / 4 \Leftrightarrow \boxed{\pi \approx 4\, n_{cible}/n_{tot}}$$



Estimation de π avec un tirage au sort

$$\pi \approx 4\, n_{cible} / n_{tot}$$

Résolution avec une boucle for :

```
ntot = 1000
ncible = 0
for i=1:ntot
    x = rand(); % nombre entre 0 et 1
    y = rand(); % nombre entre 0 et 1
    r = sqrt(x^2 + y^2); % distance à l'origine
    if r < 1
        ncible = ncible + 1;
    end
end
approx = 4*ncible/ntot;
fprintf("Approximation de pi : %f.\n",approx)
```

Estimation de π avec un tirage au sort

Résolution avec une boucle while, avec **critère de convergence** (on continue tant que le résultat change de + d'un millième) :

```
ntot = 0; % nombre de "lancers"
ncible = 0; % nombre de "touchers"
approx = 0; % approximation de pi
delta = Inf; % changement par rapport à la dernière approx
while delta > 0.001 || ntot < 20
      x = rand(); % nombre entre 0 et 1
      y = rand(); % nombre entre 0 et 1
      r = sqrt(x^2 + y^2); % distance à l'origine
      if r < 1
            ncible = ncible + 1;
      end
      ntot = ntot + 1;
      new_approx = 4*ncible/ntot;
      delta = abs(approx - new_approx);
      approx = new_approx;
end
fprintf("Il a fallu %d itérations.\n",ntot)
fprintf("Approximation de pi : %f.\n",approx)
```

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 ?

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

Un autre exemple : marche aléatoire

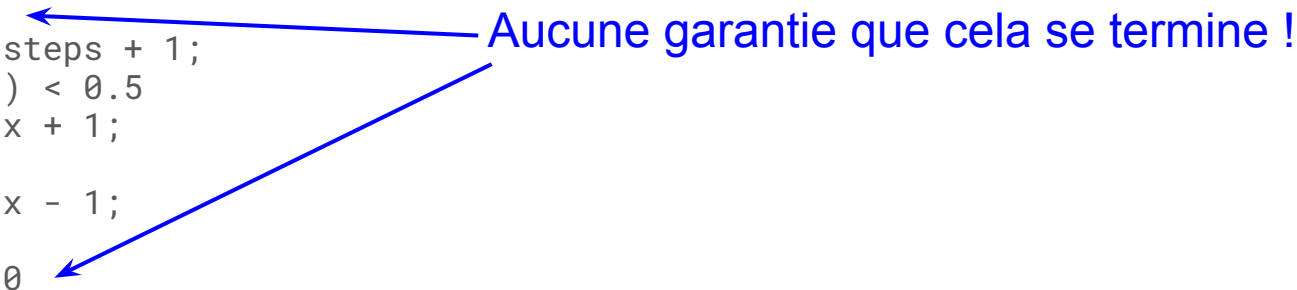
Code pour **une** marche aléatoire. Le code suivant contient un danger :

```
fini = false;
x = 0; % position
steps = 0;
while ~fini
    steps = steps + 1;
    if rand() < 0.5
        x = x + 1;
    else
        x = x - 1;
    end
    if x == 0
        fini = true;
    end
end
disp(steps)
```

Un autre exemple : marche aléatoire

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

```
fini = false;
x = 0; % position
steps = 0;
while ~fini
    steps = steps + 1;
    if rand() < 0.5
        x = x + 1;
    else
        x = x - 1;
    end
    if x == 0
        fini = true;
    end
end
disp(steps)
```



Un autre exemple : marche aléatoire

Code pour **une** marche aléatoire avec nombre maximum d'étapes (ici 10 pour répondre à la question initiale).

```
fini = false;
x = 0;
steps = 0;
max_steps = 10;
while ~fini && steps < max_steps
    steps = steps + 1;
    if rand() < 0.5
        x = x + 1;
    else
        x = x - 1;
    endif
    if x == 0
        fini = true;
    endif
end
disp(steps)
```

Un autre exemple : marche aléatoire

Simulation du taux de marcheurs qui repassent par l'origine en moins de 10 étapes:

```
N = 1000; % nombre de répétitions de l'expérience
n = 0; % nombre de fois où un marcheur est repassé par zéro en moins de 10 étapes
for i=1:N
    fini = false;
    x = 0;
    steps = 0;
    max_steps = 10;
    while ~fini && steps < max_steps
        steps = steps + 1;
        if rand() < 0.5
            x = x + 1;
        else
            x = x - 1;
        end
        if x == 0
            fini = true;
        end
    end
    if fini
        n = n + 1;
    end
end
fprintf("Le taux de retour en 10 étapes est %f.\n", n/N);
```

Un autre exemple : marche aléatoire

NB : **encapsulation** du code dans une **fonction** !

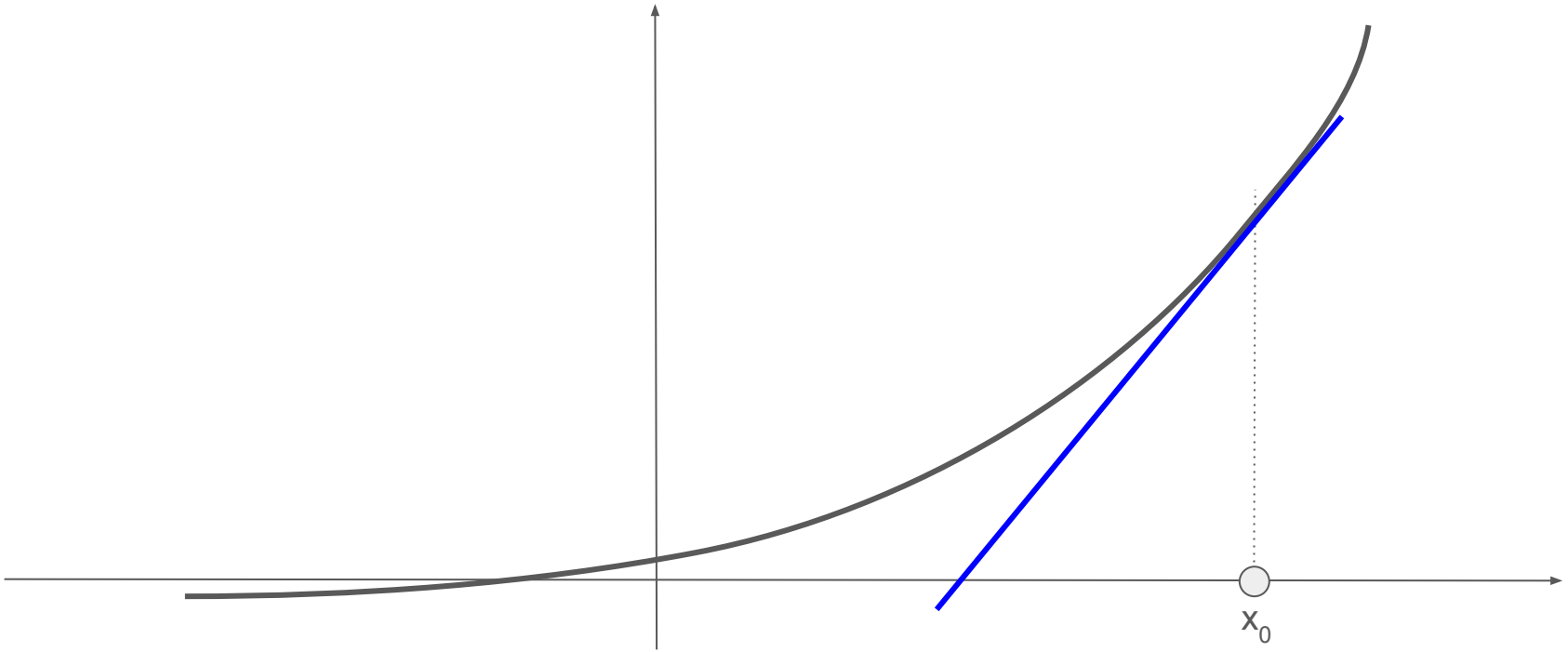
```
N = 1000;
n = 0;
for i=1:N
    fini = simule_un_marcheur();
    if fini
        n = n + 1;
    end
end
fprintf("Le taux de retour en 10 étapes est %f.\n", n/N);
```

Méthode de Newton-Raphson (pour TP du jour)

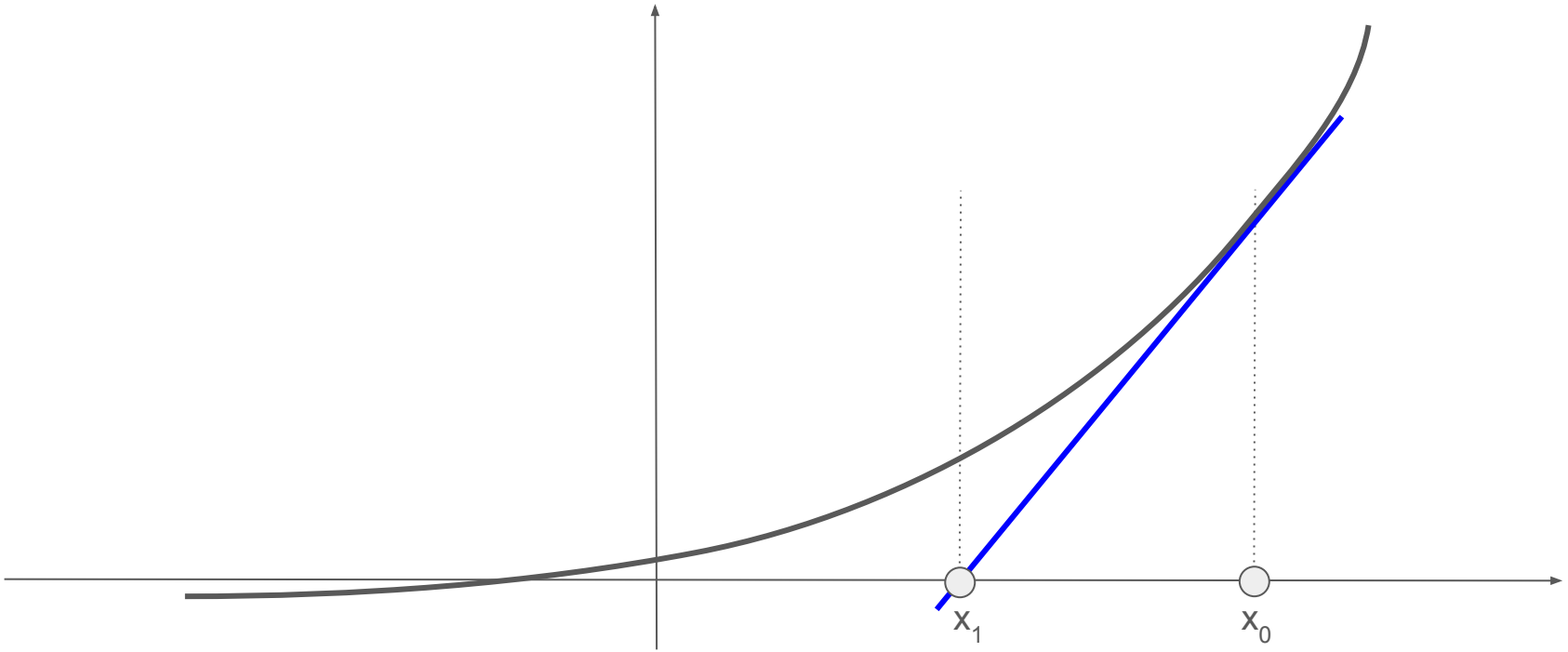
- Algorithme pour approximer le zéro d'une fonction : $f(x) = 0$.
- Approximation locale de f par sa tangente :
  $f(x) \approx f(x_0) + f'(x_0)\,(x - x_0) = f(x_0) + f'(x_0)\,x - f'(x_0)x_0 = 0$

  Par conséquent : $x = x_0 - f(x_0) \,/\, f'(x_0)$
- Résumé : partant d'un point x_0, on peut appliquer l'algorithme jusqu'à ce que l'approximation obtenue ne change pas trop.
- NB : on ignore ici les hypothèses sur f pour que l'algorithme converge.

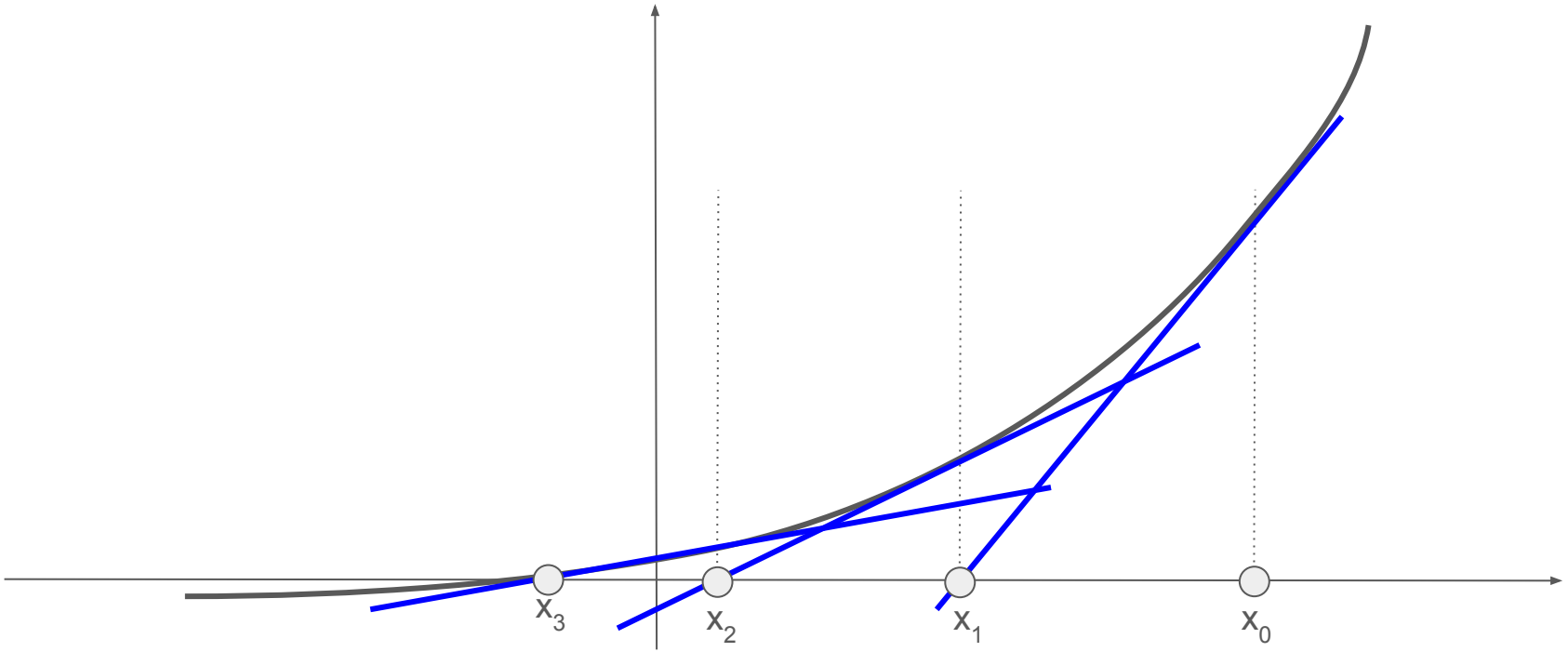
Méthode de Newton-Raphson (pour TP du jour)



Méthode de Newton-Raphson (pour TP du jour)



Méthode de Newton-Raphson (pour TP du jour)



Méthode de Newton-Raphson (pour TP du jour)

x_3 est une approximation du zéro exact.

Dans cet exemple, si x_i passe “sous” l’axe des x, x_{i+1} sera sur sa droite en raison de la forme f(x)/f’(x).