Synthèse au 3 novembre

- retour sur les fonctions
- boucle avec test en tête
- boucle `while` avec compteur
- retour sur affectation

Fonctions

Nous avons vu:

- fonctions comme sous-programme (=« bloc d'instructions nommé »)
- paramétrisation
- outil très important dans la conception:
  - modularité
  - tests unitaires

Il nous reste à voir:

- fonction python comme approximation de la fonction mathématique (qui fournit un résultat)
- instruction `return`
- notion d'effet de bord

Sorbonne Nouvelle

Principe de la boucle

```
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
```

Principe de la boucle

```
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
```

```
marche_droite(50,100)
marche_droite(50,100)
marche_droite(50,100)
```

Sorbonne Nouvelle

Principe de la boucle

```
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
# 1 marche:
left(90)
fd(50)
right(90)
fd(100)
```

```
marche_droite(50,100)
marche_droite(50,100)
marche_droite(50,100)
```

```
répeter :
    marche_droite(50,100)
```

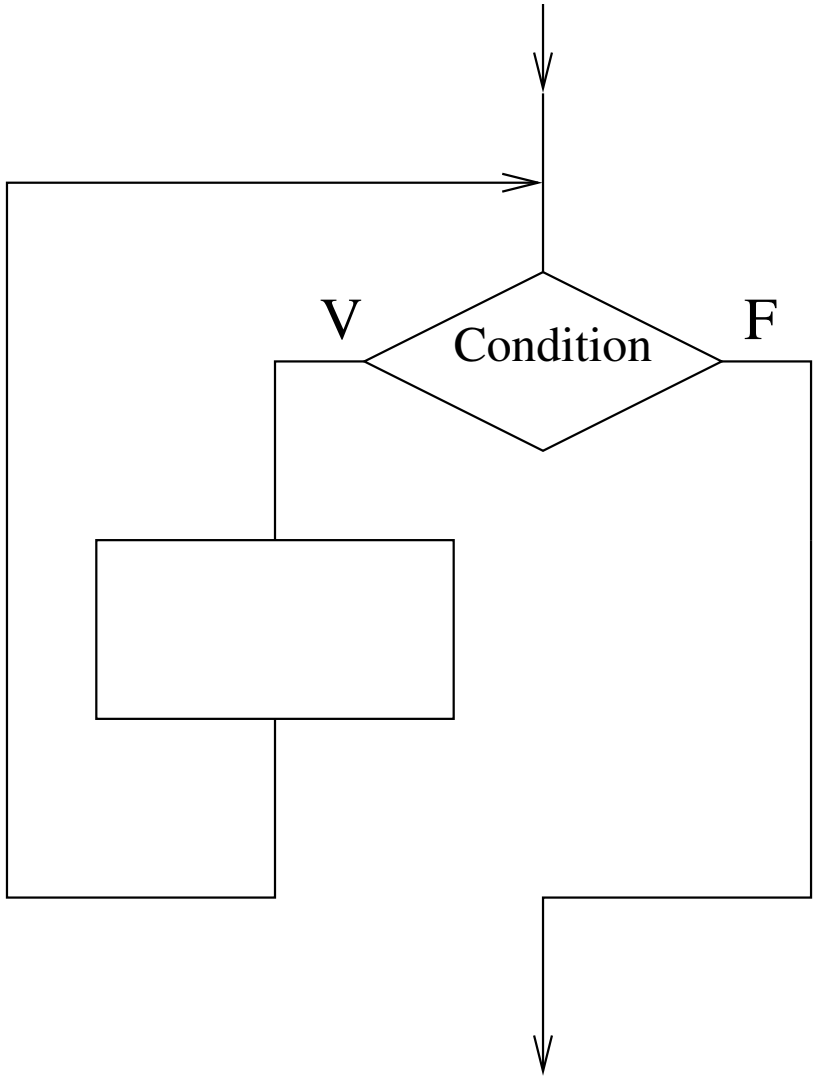
Sorbonne Nouvelle

Contrôle de la boucle

On ne veut pas répéter indéfiniment: il faut donc établir une “condition d’arrêt”

Contrôle de la boucle

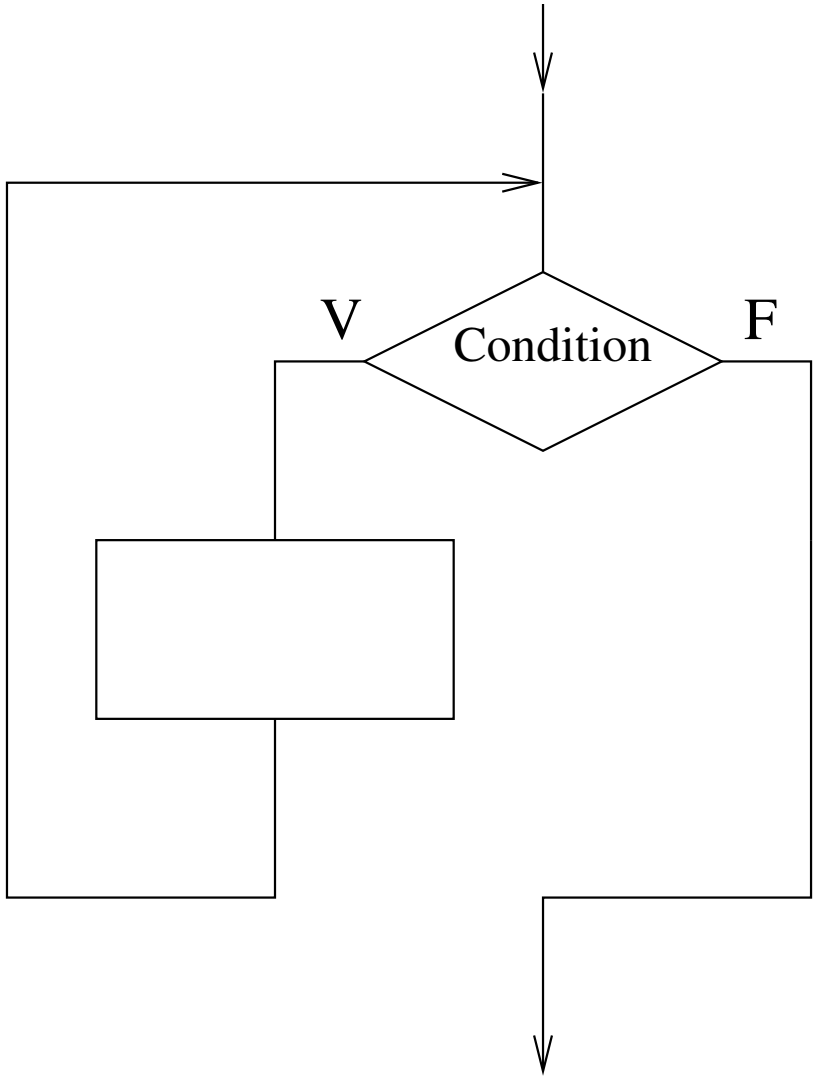
On ne veut pas répéter indéfiniment: il faut donc établir une "condition d'arrêt"



Sorbonne Nouvelle

Contrôle de la boucle

On ne veut pas répéter indéfiniment: il faut donc établir une "condition d'arrêt"



Boucle « avec test en tête »

Mise en œuvre

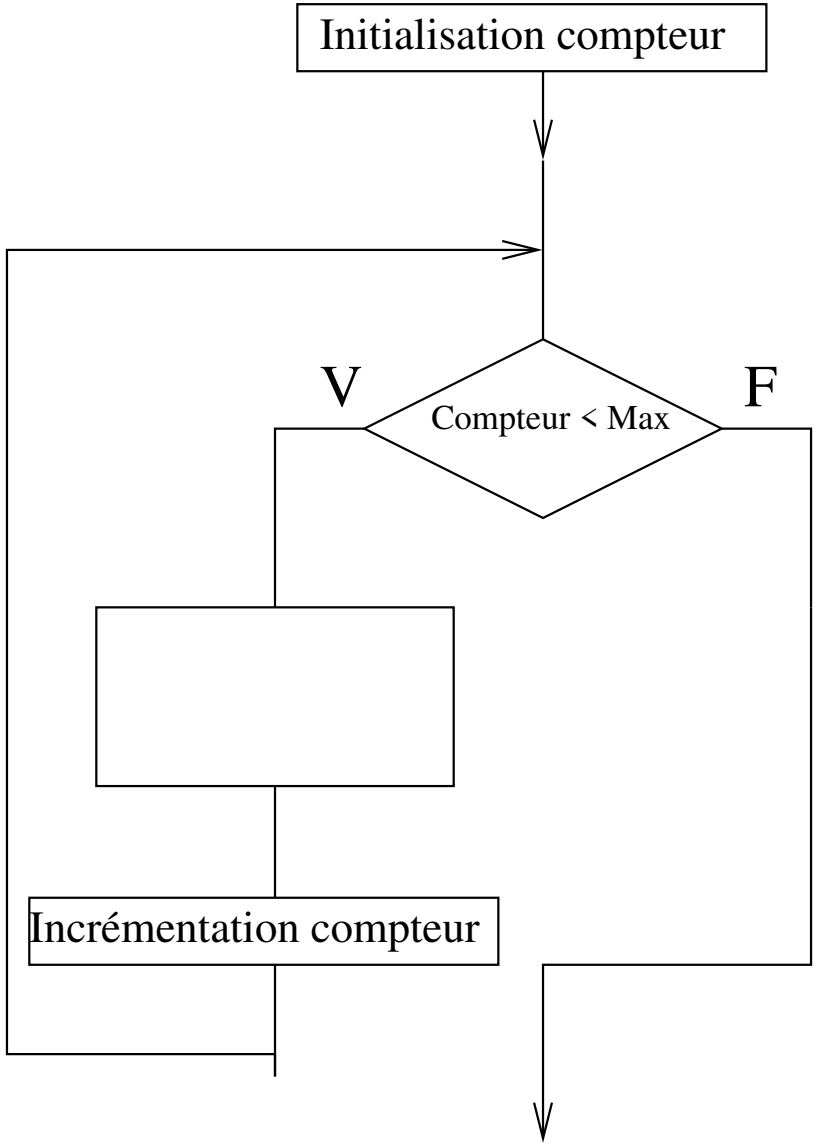
```
While <condition>:
    <bloc d'instructions>
```

Mise en œuvre

```
While <condition>:
    <bloc d'instructions>

tant que {les oignons ne sont pas translucides}:
    remuer la préparation
```

Boucle avec compteur



Exemple

```
c = 0
while (c < 4):
    forward(100)
    right(90)
    c = c + 1
```

Affectation

```
long = 100
coef = 1.2
```

Affectation

```
long = 100
coef = 1.2
```

- Première affectation:

→ réservation d'une « case mémoire »

→ détermination de son type (entier, réel...)

Affectation

```
long = 100
coef = 1.2
```

- Première affectation:

→ réservation d'une « case mémoire »

→ détermination de son type (entier, réel...)

```
long = 200
```

Affectation

```
long = 100
coef = 1.2
```

- Première affectation:

→ réservation d'une « case mémoire »

→ détermination de son type (entier, réel...)

```
long = 200
```

- Affectations suivantes:

→ remplacement de la valeur stockée par une nouvelle valeur (écrasement)

Affectation

```
long = 100
coef = 1.2
```

- Première affectation:

→ réservation d'une « case mémoire »

→ détermination de son type (entier, réel...)

```
long = 200
```

- Affectations suivantes:

→ remplacement de la valeur stockée par une nouvelle valeur (écrasement)

```
long = long * coef
```

Sorbonne Nouvelle

Affectation

```
long = 100
coef = 1.2
```

- Première affectation:

→ réservation d'une « case mémoire »

→ détermination de son type (entier, réel...)

```
long = 200
```

- Affectations suivantes:

→ remplacement de la valeur stockée par une nouvelle valeur (écrasement)

```
long = long * coef
```

- opération disymétrique:
- à gauche: nom d'une "case mémoire"
- à droite: valeur (à calculer en premier)

Sorbonne Nouvelle

Affectation

```
long = 100
coef = 1.2
```

- Première affectation:

→ réservation d'une « case mémoire »

→ détermination de son type (entier, réel...)

```
long = 200
```

- Affectations suivantes:

→ remplacement de la valeur stockée par une nouvelle valeur (écrasement)

```
long = long * coef
```

- opération disymétrique:
- à gauche: nom d'une "case mémoire"
- à droite: valeur (à calculer en premier)

C'est pourquoi alors que l'équation mathématique $x = x + 1$ n'a pas de solution, l'affectation `x ← x + 1` est parfaitement possible.

Sorbonne Nouvelle