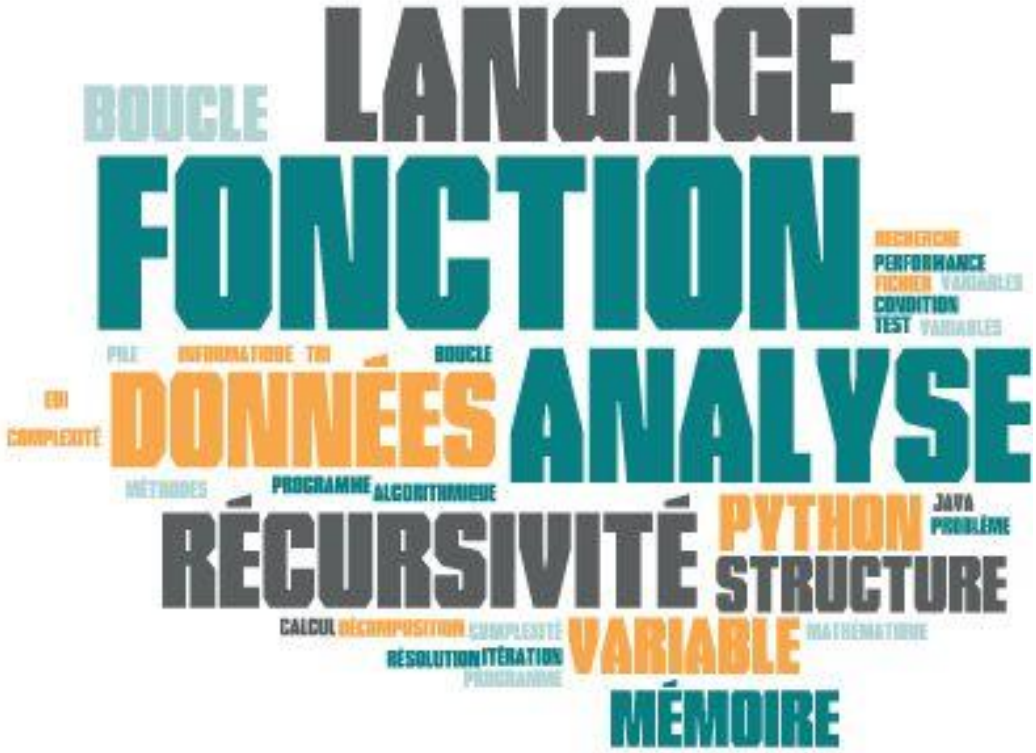
Cours : Introduction à l'algorithmique et à la programmation

PHILIPPE POLET 2025©

philippe.polet@uphf.fr

Algorithmique et Programmation?



Introduction à l'algorithmique

Un algorithme est une suite finie d'actions à réaliser pour résoudre un problème.

Nous avons des exemples d'algorithmes simples dans la vie de tous les jours:

Recette de cuisine

Notice de montage d'un meuble en kit.

Le niveau de détail et de précision des actions décrites dans un algorithme doit être de sorte que chaque action soit connue de l'exécutant et donc que l'interprétation de l'algorithme soit unique et ne prête pas à confusion.

Les types de bases couramment utilisés

Type algorithmique	Type python	Exemples de valeurs (en python)
entier	int	12 -59
réel	float	-26.987 89.39877 1.0
booléen	bool	True False
Chaîne de caractères	str	‘bonjour’ ’12’

Remarque : la valeur 12 est différente de la valeur ’12’

Notion de fonction

Comme en mathématiques, une fonction calcule une valeur à partir d'une liste de paramètres (aussi appelés arguments). Cette liste peut être vide.

En informatique, une fonction sera donc la suite d'instructions permettant de retourner une valeur à partir des paramètres d'appel.

On distingue donc deux parties :

- L'**entête** de la fonction : renseigne le nom de la fonction, la liste et le type des paramètres ainsi que le type du résultat.
- Le **corps** de la fonction : définit les variables utilisées et décrit les instructions qui permettent de calculer le résultat.

Dans la plupart des langages il faut au moins une fonction: la fonction principale, qui est exécutée au démarrage du programme.

Notion d’arbre programmatique

Un arbre programmatique permet de décrire un algorithme sous une forme graphique.

Un arbre programmatique doit servir de support à la réflexion mais également comme un outil de communication.

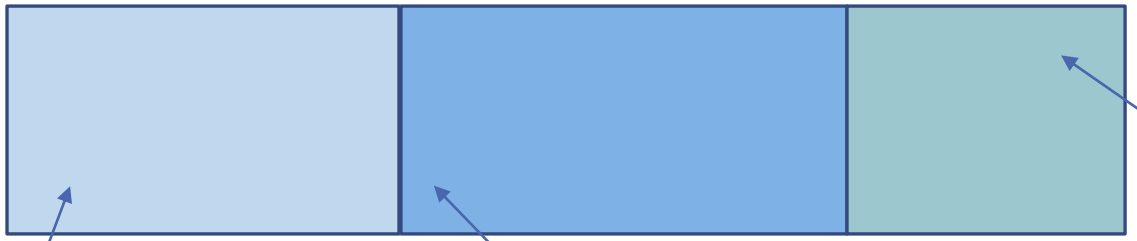
Un arbre programmatique est indépendant d’un langage. Le même arbre programmatique pourra être traduit dans différents langages.

Il est particulièrement adapté pour les langages impératifs (ADA, C, Pascal…).

Un arbre programmatique décrit les instructions d’une fonction.

Commençons par une fonction simple: une fonction qui retourne la valeur (entier) de son paramètre (entier) élevée au carré.

L’entête d’un arbre programmatique

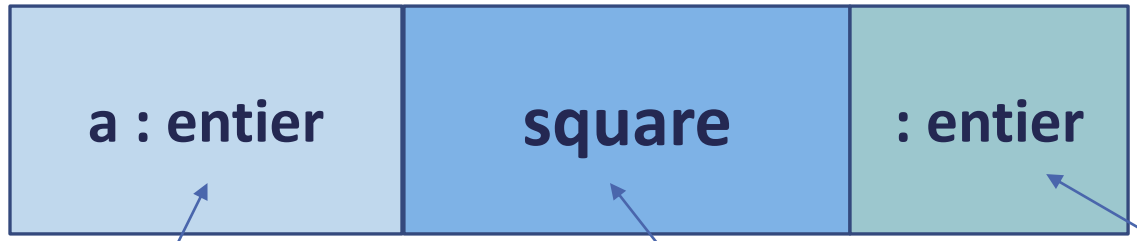


Liste des paramètres

Nom (identifiant) de la fonction

Type du résultat
Si la fonction ne retourne rien (elle ne fait juste qu’un traitement alors on met **void**)

L’entête de la fonction *square*

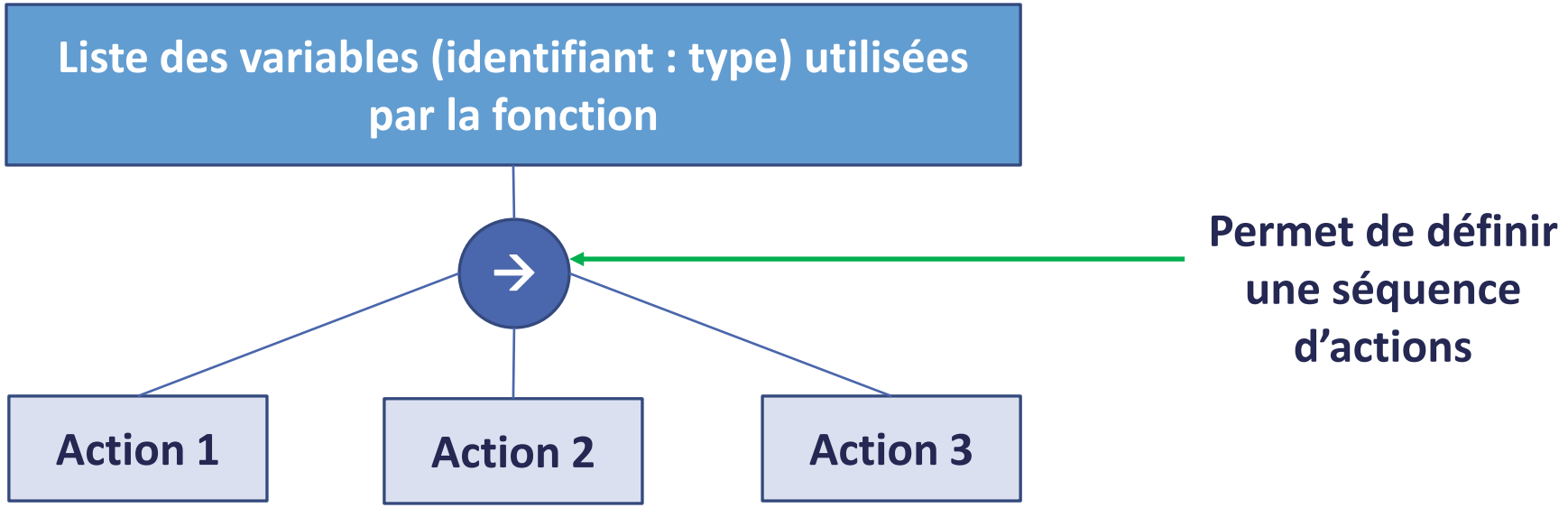


Liste des paramètres

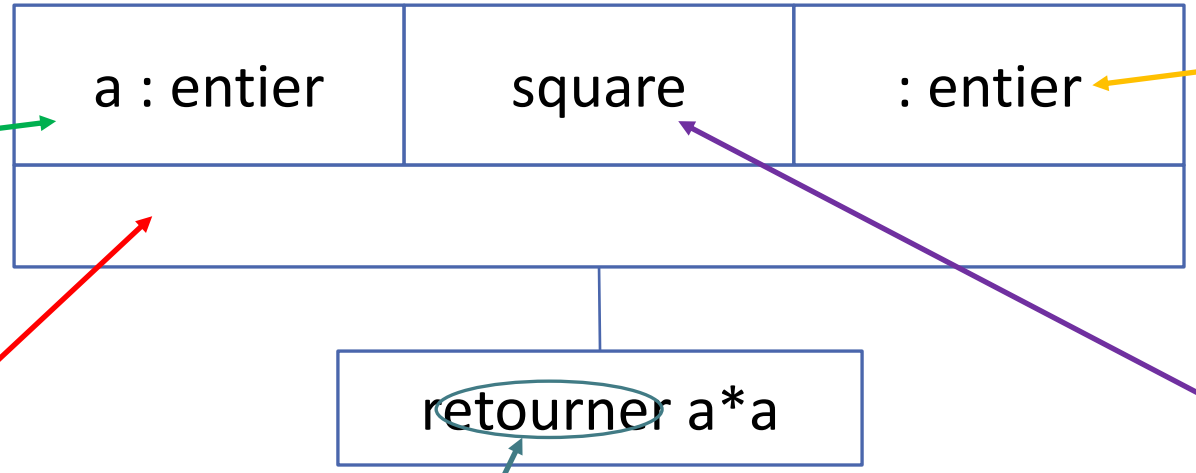
Nom (identifiant) de la fonction

Type du résultat

Le corps d’une fonction en arbre programmatique



Arbre complet de la fonction *square*



Type de la valeur calculée par la fonction

Un seul paramètre : a

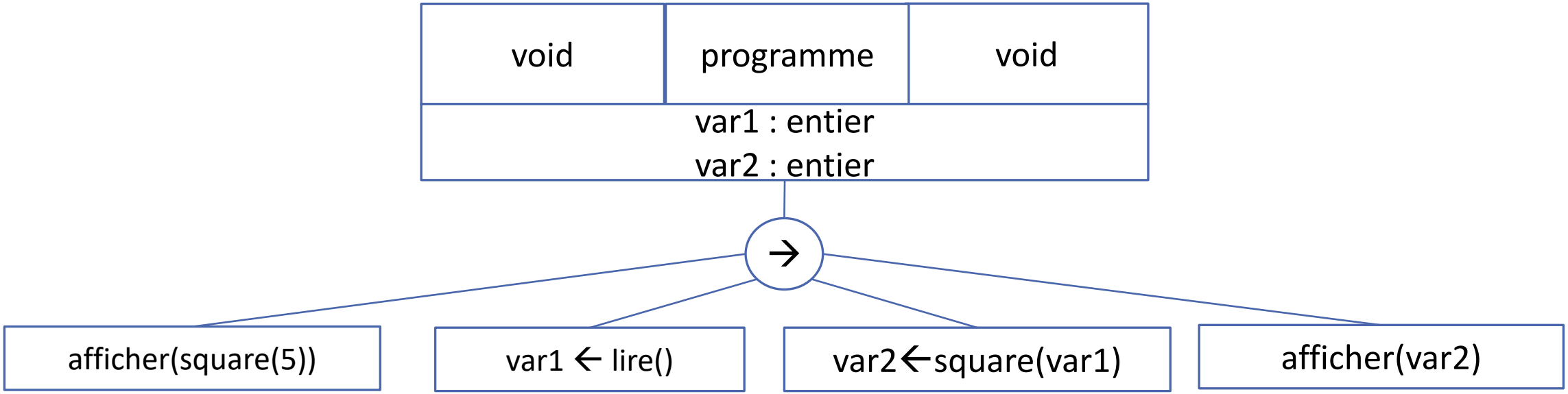
Nom de la fonction

La fonction n'utilise aucune variable

Mot réservé pour indiquer la valeur qui sera renvoyée par la fonction. Par conséquent, dès que cette instruction est terminée l'algorithme est terminé!
Même s'il y a d'autres instructions après, elles ne seront pas exécutées.

Programme principal

On veut un programme qui affiche 5 au carré, puis demande à l'utilisateur de saisir une valeur entière et affiche le carré de cette valeur.



Traduction d'un arbre programmatique en python

Les variables ne sont pas déclarées en python

`Le type est automatiquement attribué`

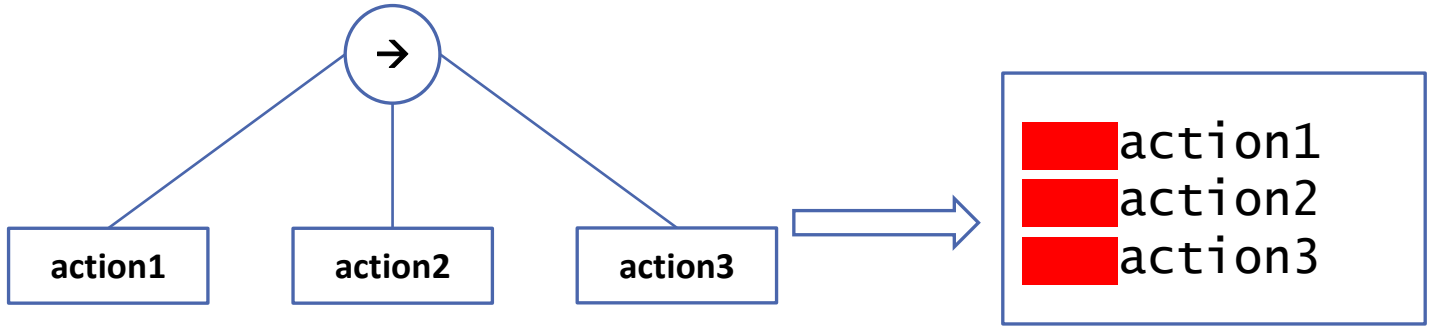
L'opérateur d'affectation (←) est le `=` par exemple `v = 5`

Les fonctions sont déclarées de la manière suivante :

```
def nomDeLaFonction(nomDesParametres):
    debut instructions
```

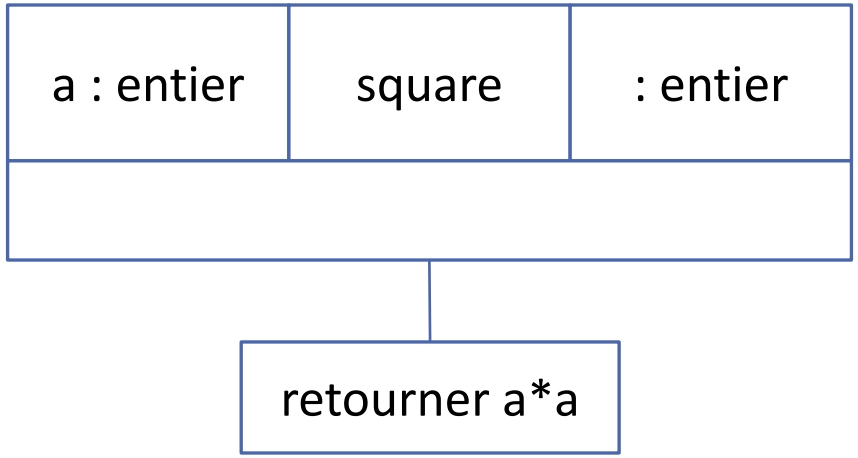
`Attention, il faut indenter (espace ou tabulation)`

La séquence d'actions : les actions connectées à un nœud sont écrites au même niveau d'indentation:



Traduction de la fonction *square*

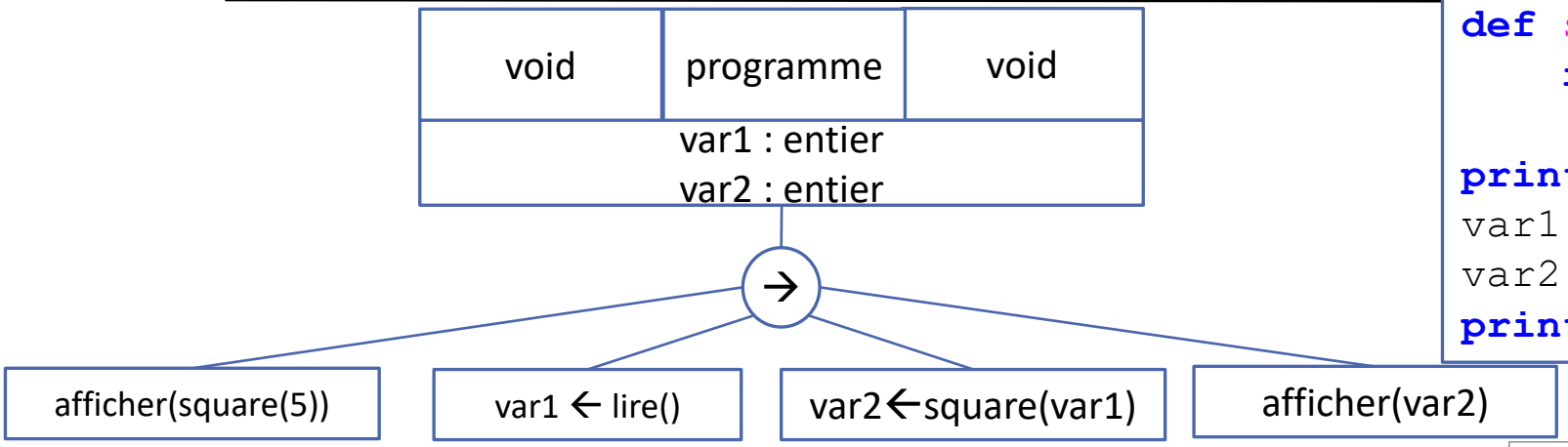
Arbre Programmatique



Python

```
def square(a):
    return a*a
```

Traduction du programme principal

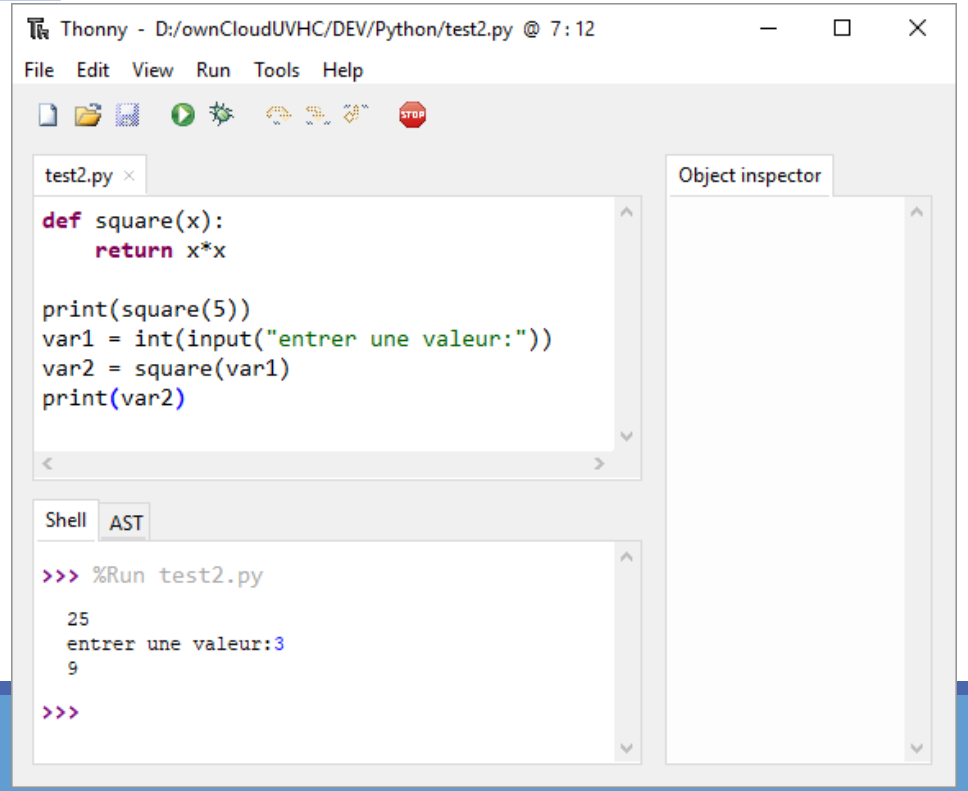



```
def square(x):
    return x*x

print(square(5))
var1 = int(input("entrer une valeur:"))
var2 = square(var1)
print(var2)
```

En python le programme principal commence à la première ligne qui n'est pas dans une fonction

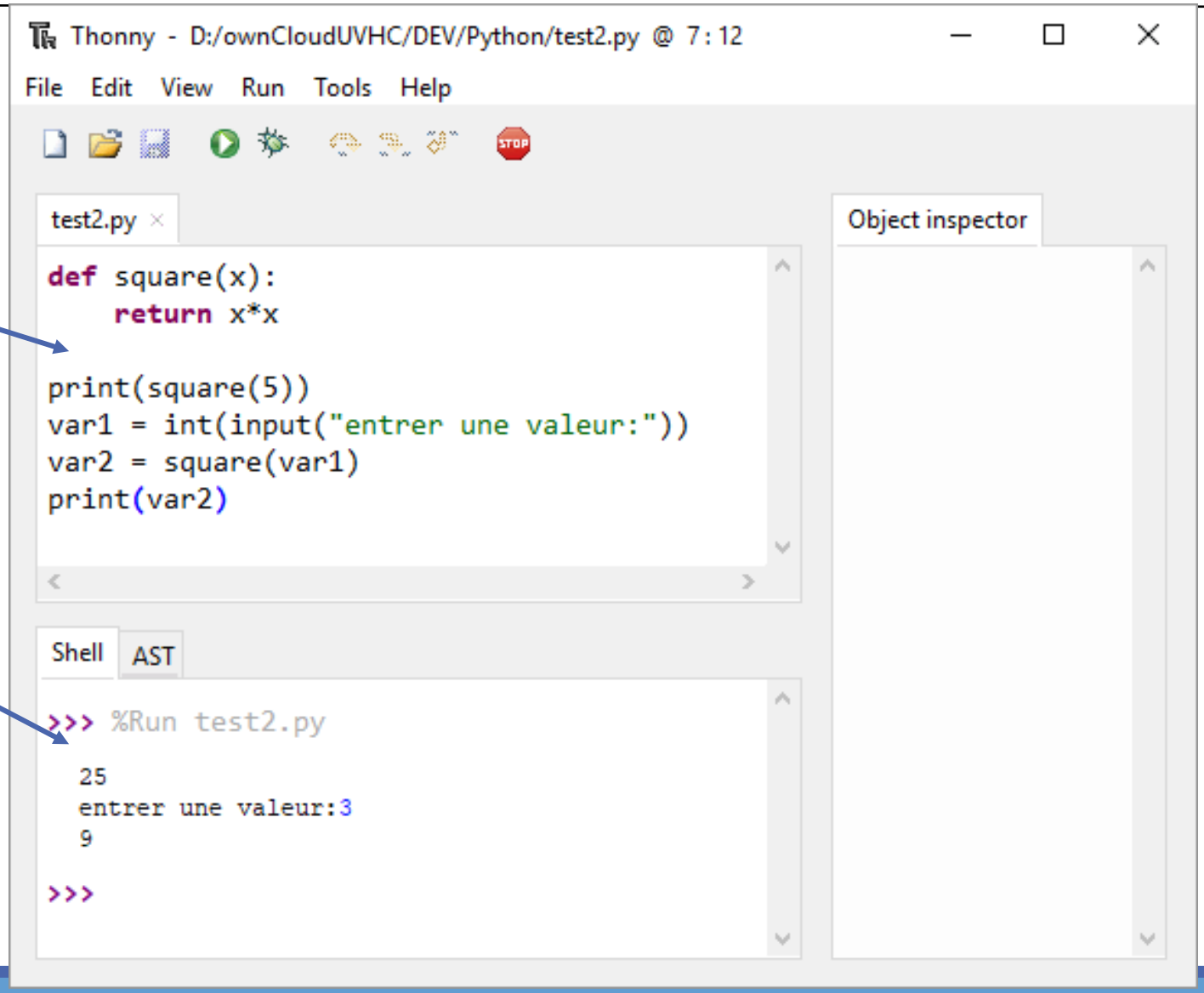
print est la fonction en python qui permet d'afficher
input est la fonction en python qui permet de lire. On peut mettre un message en paramètre. Par défaut, input renvoie une chaîne de caractères qu'on traduit en int (entier) dans notre exemple



Environnement de programmation

Zone d'édition du script

Zone d'exécution du script



Un peu de logique : les booléens (bool en python)

Une expression booléenne ne peut admettre que 2 valeurs VRAI ou FAUX (en python True ou False)

Opérations simples sur les booléens :

L'opérateur NON (en python not)

NON VRAI = FAUX , NON FAUX = VRAI

L'opérateur ET (en python and)

a	b	a ET b
VRAI	VRAI	VRAI
VRAI	FAUX	FAUX
FAUX	VRAI	FAUX
FAUX	FAUX	FAUX

L'opérateur OU (en python or)

a	b	a OU b
VRAI	VRAI	VRAI
VRAI	FAUX	VRAI
FAUX	VRAI	VRAI
FAUX	FAUX	FAUX

Opérateurs retournant une valeur booléenne

Les opérateurs de comparaison renvoient une valeur booléenne:

Soit **a** une variable de type numérique (int, float, etc…) .

L'expression $a \leq 10$ est de type booléen. Ainsi, si a vaut 5, alors l'expression vaut VRAI.

Les opérateurs de comparaison

Notation algorithmique	Notation en python	Descriptif
$a = b$	$a == b$	Retourne VRAI si a est égal à b (en python il s'agit d'un double égal)
$a \neq b$	$a\ != b$	Retourne VRAI si a est différent de b
$a < b$	$a < b$	Retourne VRAI si a est strictement plus petit que b
$a \leq b$	$a <= b$	Retourne VRAI si a est plus petit ou égal à b
$a > b$	$a > b$	Retourne VRAI si a est strictement plus grand que b
$a \geq b$	$a >= b$	Retourne VRAI si a est plus grand ou égal à b

Traitement conditionnel

Il est souvent nécessaire de réaliser des traitements particuliers (suite d'actions) selon certaines conditions.

C'est ce que nous permet de faire la structure de contrôle « Si Alors Sinon » (en anglais : If Then Else, *ite*)

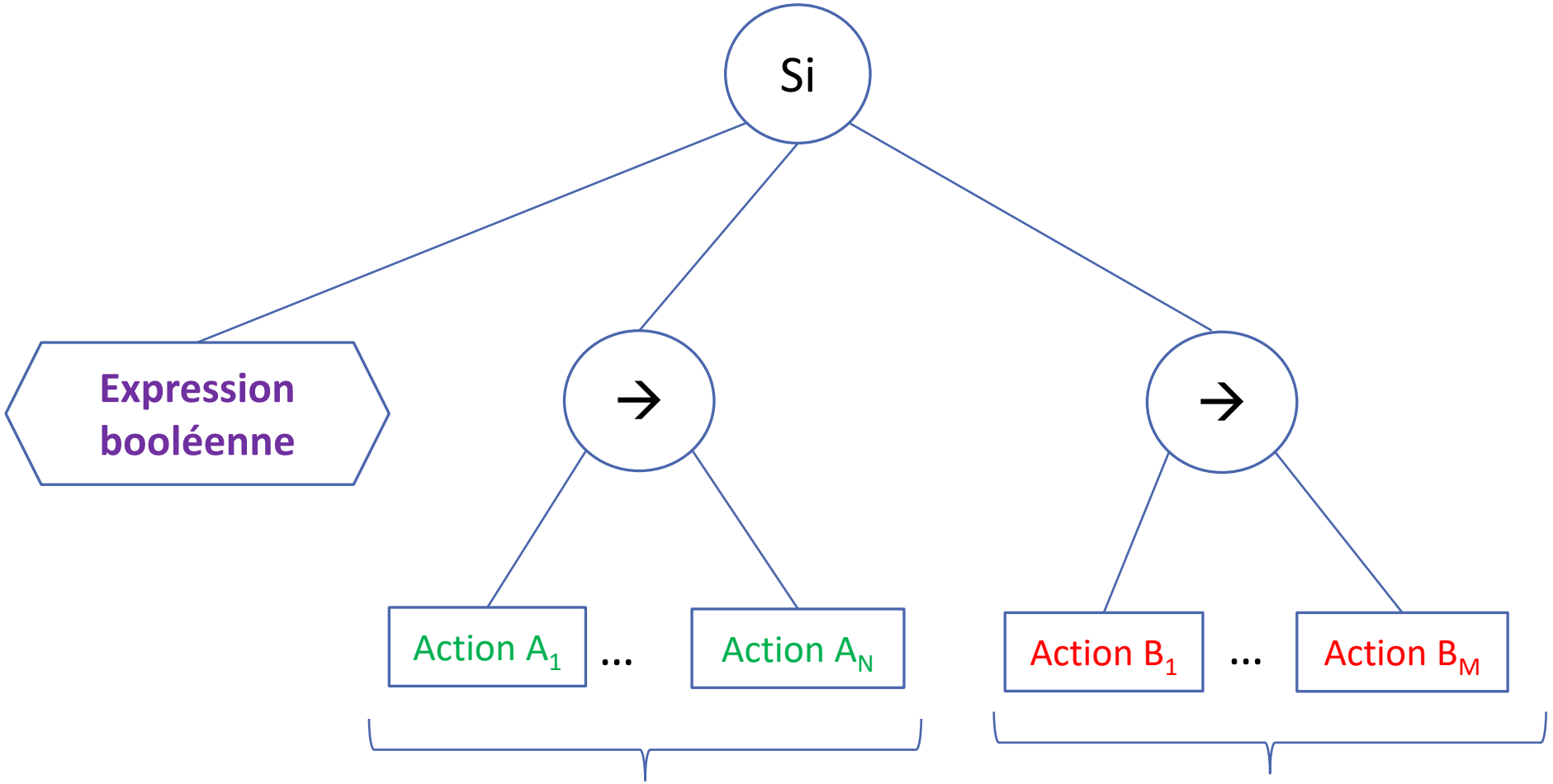
Le principe :

Si expression booléenne vraie

Alors on réalise le traitement A

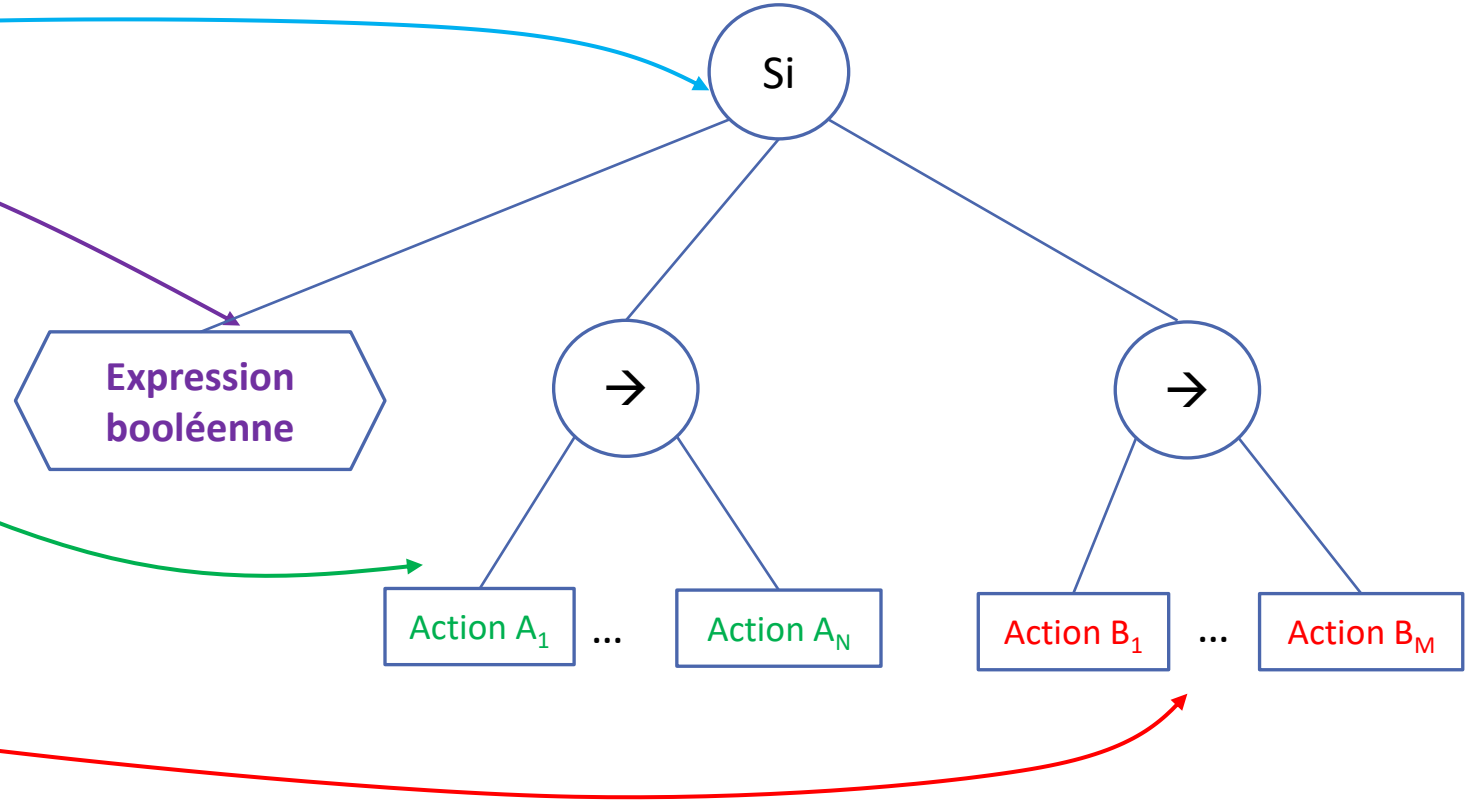
Sinon on réalise le traitement B

Représentation en arbre programmatique



Code en Python

```
if(expression) :
    actionA1
    ...
    actionAn
else :
    actionB1;
    ...
    actionBm;
```



A noter :

1. l'expression booléenne doit être placée entre **parenthèses**
2. Pas de mot clef pour le « alors »
3. Mot clef pour le bloc du « sinon » est **else**

ne pas oublier les : et INDENTER

Exemple d'application

On souhaite écrire une fonction qui retourne l'état de l'eau en fonction de la température.

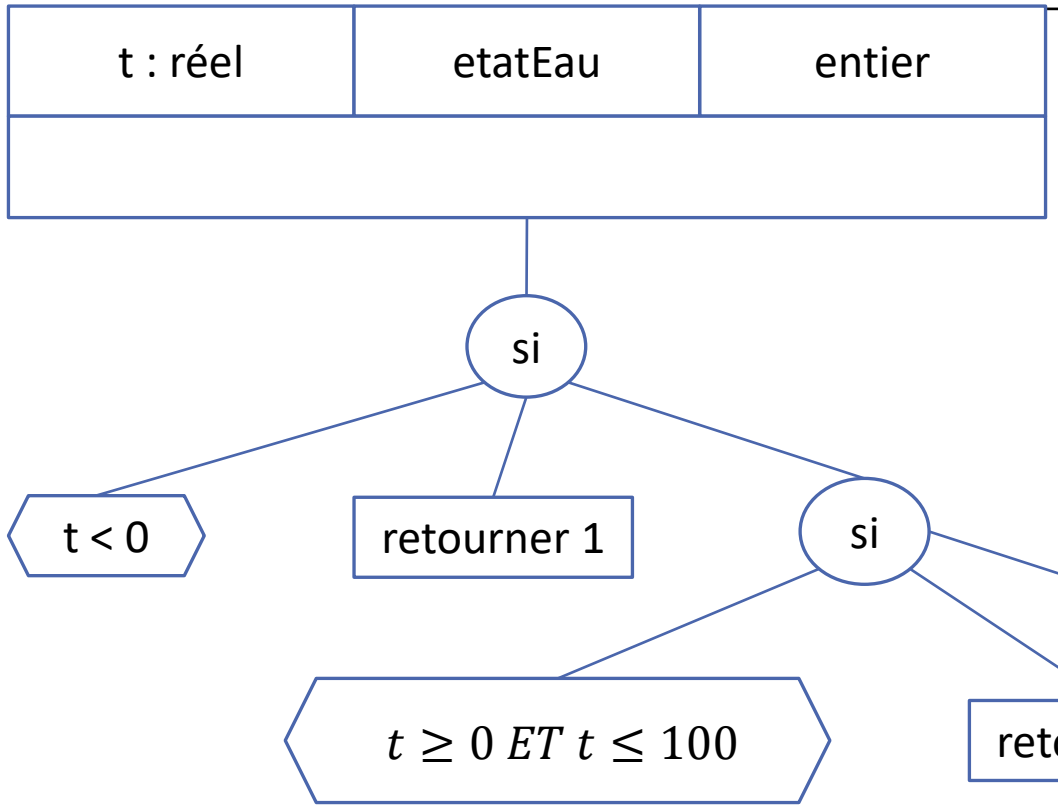
La température sera donnée sous la forme d'un réel.

L'état sera une valeur entière telle que:

- 1 correspond à l'état solide,
- 2 correspond à l'état liquide,
- 3 correspond à l'état gazeux.

Quand la température est inférieure à 0°C, l'eau est à l'état solide. De 0 °C à 100°C, l'eau est à l'état liquide, et au-delà elle est à l'état gazeux.

Arbre Programmatique et code Python




```
def etatEau(temp):
    if(temp<0):
        return 1
    else :
        if((temp>=0) and (temp<=100)):
            return 2
        else:
            return 3
```

On remarquera l'imbrication des structures de contrôles. En arbre programmatique, cette imbrication apparaît naturellement. En programmation, c'est le codeur qui doit faire un **effort** pour la rendre **visible**. Il est important d'adopter dès le début des règles d'écriture de code et notamment s'appliquer à **indenter** son code quelque soit le langage.

Exemple de problème

On désire écrire un algorithme d'une fonction qui admet en paramètre une valeur réelle (x) et une valeur entière positive (n) et qui retourne x^n.

Analyse :

Cas particulier $n = 0 \Rightarrow x^n = 1$

$$x^1 = x$$
$$x^2 = x * x$$
$$x^3 = x * x * x$$
$$\dots$$
$$x^n = x * \cdots * x$$

Pour calculer x^non peut multiplier $x\ par\ lui\ même, n-1\ fois$

Répétition conditionnelle

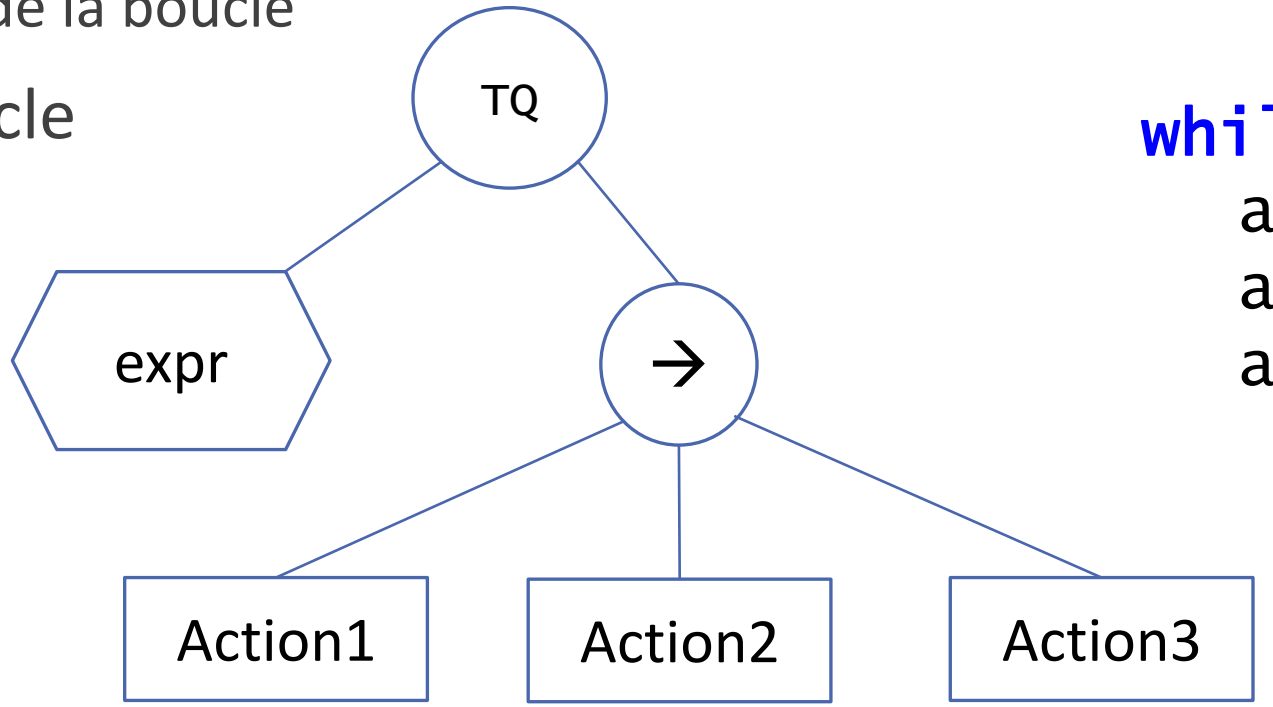
Principe : répéter un traitement tant qu'une expression est vraie

En début de cycle on évalue une expression booléenne:

Si elle vaut VRAI
- On réalise le traitement de la boucle

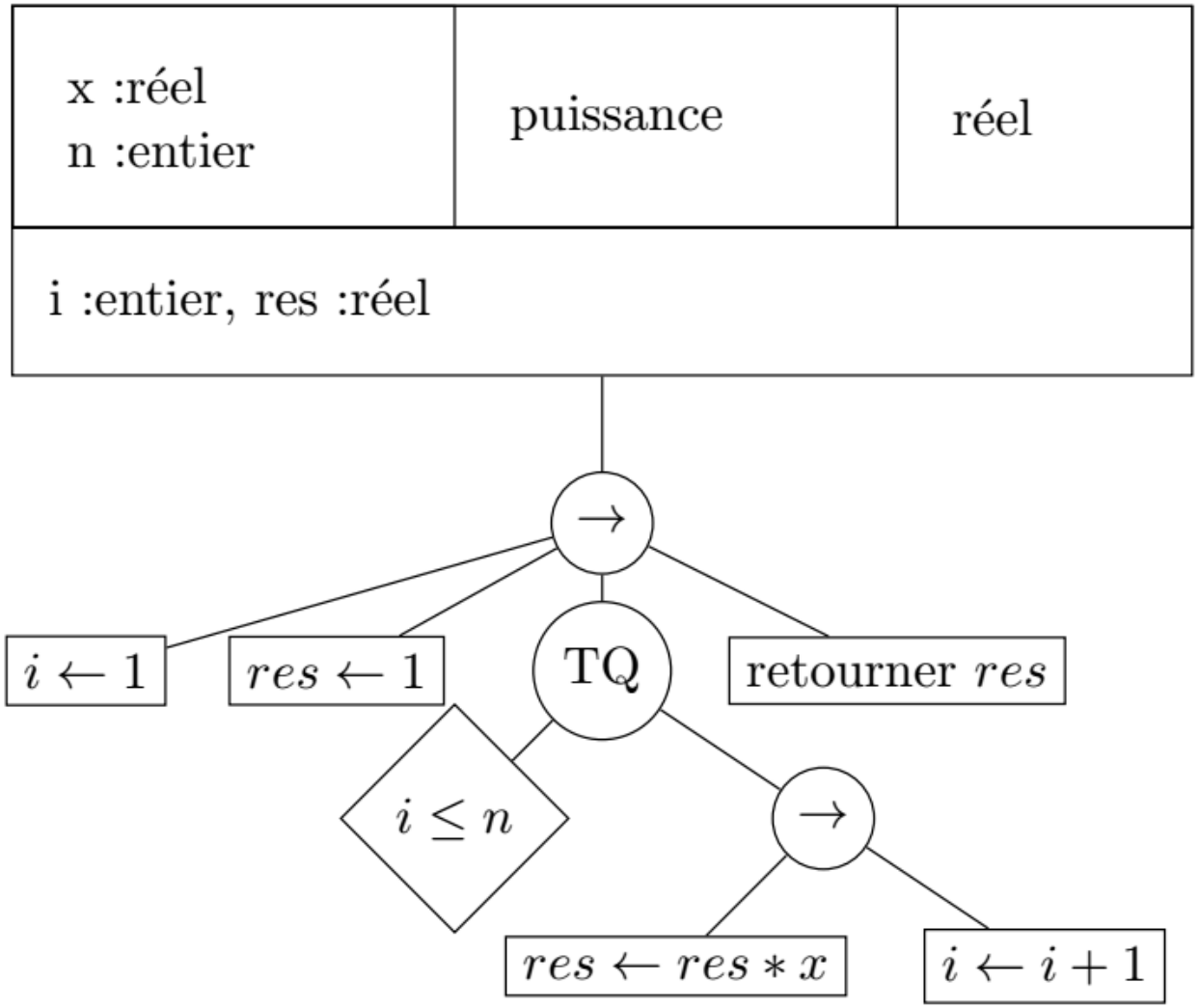
Sinon on sort de la boucle

La boucle « tant que »




```
while(expr):
    action1
    action2
    action3
```

La fonction puissance avec une boucle *tant que*




```
def puissance(x,n):
    i = 1
    res = 1
    while(i<=n):
        res = res * x
        i = i + 1
    return res
```