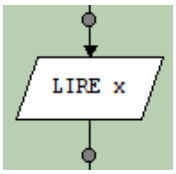
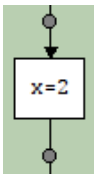
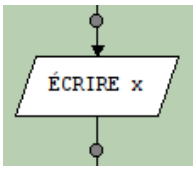
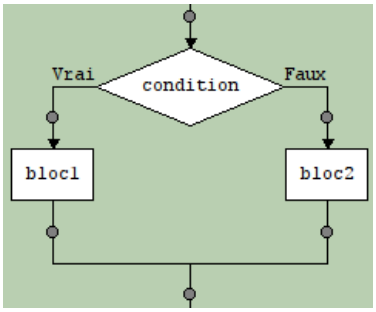
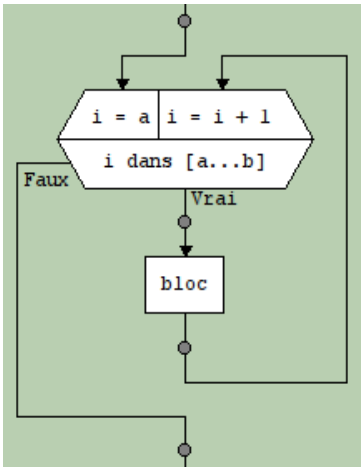
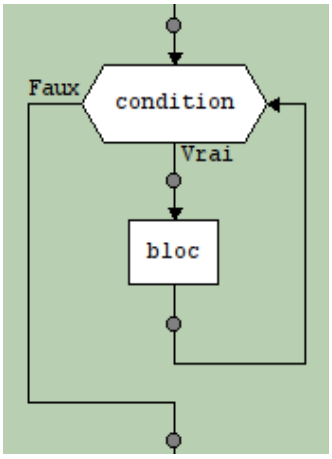
	AlgoBox	**LARP**	**Python**
Remarques	Il faut préalablement déclarer les variables avec le bouton + Déclarer nouvelle variable `x EST_DU_TYPE NOMBRE`	LARP offre beaucoup de modèles prédéfinis ; en survolant les modèles dans la colonne de gauche, il affiche leur nom.	
Saisir dans *la variable x*	+ Ajouter LIRE variable `LIRE x`		`x=input("x=")`
La variable x **prend** ***la valeur*** *2*	+ Affecter valeur à variable `x PREND_LA_VALEUR 2`		`x=2`
Afficher *la valeur x*	+ Ajouter AFFICHER Variable `AFFICHER x`		`print x`

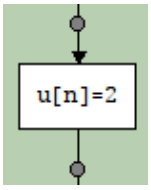
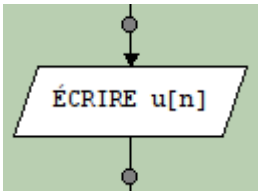
Si *condition* **alors** *bloc1* **sinon** *bloc2*	+ Ajouter SI...ALORS
	`SI (condition) ALORS`
	`  DEBUT_SI`
	`  bloc1`
	`  FIN_SI`
	`  SINON`
	`  DEBUT_SINON`
	`  bloc2`
	`  FIN_SINON`
			`if (condition):`
			`  bloc1`
			`else:`
			`  bloc2`

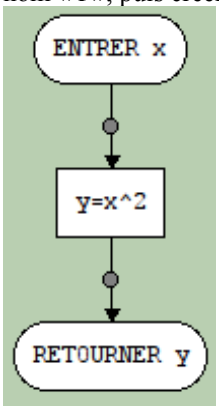
Pour *i* **variant de** *a* **à** *b* **faire** *bloc*	+ Ajouter POUR...DE...a
	`POUR i ALLANT_DE a A b`
	`  DEBUT_POUR`
	`  bloc`
	`  FIN_POUR`
			`for i in range(a,b+1):`
			`  bloc`
			
			Avec le module liste.py - ajouter dans le préambule :
			`from liste import *`
			
			`for i in entiers(a,b):`
			`  bloc`

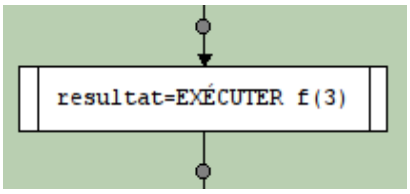
Tant que *condition* **faire** *bloc*	+ Ajouter TANT QUE...
	`TANT_QUE (condition)`
	`FAIRE`
	`  DEBUT_TANT_QUE`
	`  bloc`
	`  FIN_TANT_QUE`
			`while (condition):`
			`  bloc`

	AlgoBox	**LARP**	**Python**
Travailler avec des listes ou des suites			

Remarques	Il faut préalablement déclarer la variable u de type liste
	+ Déclarer nouvelle variable
	`u EST_DU_TYPE LISTE`
		LARP dispose par défaut de la structure de variable liste.	En python, il faut « créer » la liste u avant de s'en servir, car ce n'est pas un type de base.
			On peut utiliser le module liste.py, en ajoutant dans le préambule :
			`from liste import *`
			puis on déclare une suite :
			`u=liste()`

La variable u[n] **prend *la valeur*** *2*	+ Affecter valeur à variable `u[n] PREND_LA_VALEUR 2`		`u[n]=2`
Afficher *la valeur u[n]*	+ Ajouter LIRE variable `AFFICHER u[n]`		`print u[n]`
Travailler avec une fonction			

Définir la fonction *f d'expression* $f(x)=x^2$	Dans l'onglet « Utiliser une fonction numérique », cocher la case « Utiliser une fonction », puis définir la fonction
	`F1(x)=pow(x,2)`
	
	Remarque : la notation `x^2` n'est pas reconnue par algobox (ni `x**2`).
		Cliquer -droit dans la colonne de gauche, partie « Navigateur » : sélectionner « Nouveau module (organigramme) », et lui donner pour nom « f », puis créer :
		 
		Remarque : la notation `x**2` n'est pas reconnue par LARP
			`def f(x):`
			`  y=x**2`
			`  return y`
			
			Python utilise la notation `x**2` pour la puissance (plutôt que `x^2`). On peut contourner cela en utilisant le module math : ajouter dans le préambule
			`from math import *`
			on peut alors utiliser la commande
			`y=pow(x,2)`

La variable resultat **prend *la valeur*** *f(3)*	+ Affecter valeur à variable
	`resultat`
	`PREND_LA_VALEUR F1(3)`
			`resultat=f(3)`

Remarques générales sur la représentation des données :
Les variables contiennent, principalement, une valeur « entière » ou « flottante » ; ces deux représentations sont limitées en capacité (ex : plus grand « entier » représentable) et en précision (ex : plus petite valeur absolue « flottante » représentable). De ce point de vue, python semble présenter les meilleures performances.
Attention cependant à la division, qui peut être « entière » ou « flottante » : par exemple, `7` est un entier pour python, tandis que `7.0` est un flottant. Ainsi l'instruction `print 7/2` retournera `3`, alors que l'instruction `print 7.0/2` retournera `3.5`