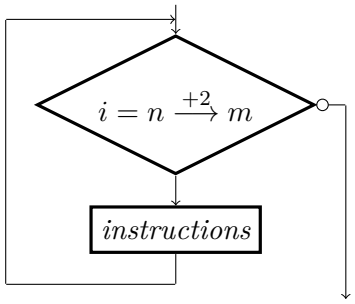
Nous réutiliserons dans ce TP les instructions et les notions vues aux TP1 et TP2 auxquelles nous ajouterons les boucles POUR.

I Structure de boucles

On souhaite répéter une même (ou similaire) instruction un nombre de fois prédéfini. Pour ce faire, on peut utiliser la structure de boucle POUR.

POUR i **ALLANT DE** n **À** m **(pas** $+2$**) FAIRE**
    les instructions sont effectuées
FIN-POUR
Le reste du programme continue.



Sous python, la structure de boucle POUR se présente ainsi

```
for i in range (n,m,2):     # i prendra les valeurs entieres de 2 en 2 entre n
    et m-1.
   bloc d instructions      # bloc repete n-m fois
instructions suivantes      # instructions a l'exterieur de la boucle
```

Par défaut la valeur initiale est 0 et le pas est 1, ainsi `range(n)` correspond à `range(0,n,1)`.

`range(n)` correspond à l'ensemble {0 ;1 ;2 ; ... n-1}.

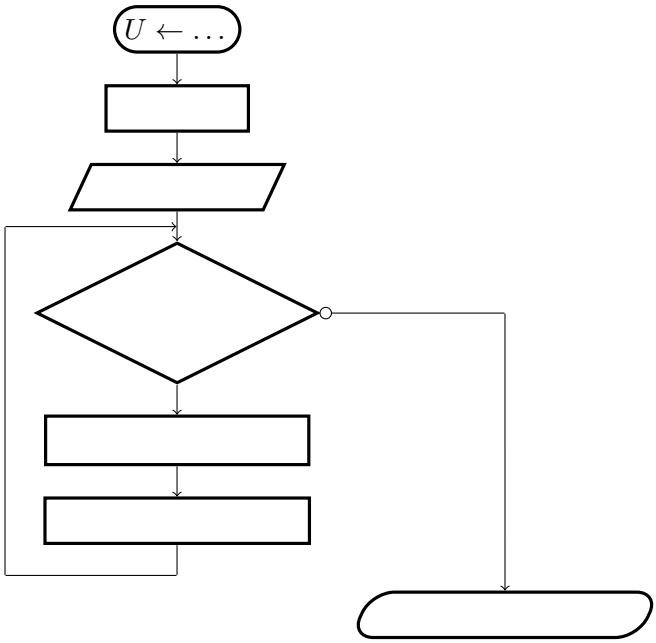
Exo 1 : À l'aide d'une boucle `POUR`, et avant de dérouler le 3è algorithme pour $N = 3$, proposer un algorithme, son organigramme et son programme `python` qui affichent

1. dix fois la phrase "je suis le meilleur / la meilleure"
2. tous les entiers entre 0 et 10
3. tous les carrés entre 0 et une valeur entière N donnée par l'utilisateur

Exo 2 : on considère la suite arithmético-géométrique (u_n) définie par $\begin{cases} u_{n+1}=\frac{3}{4}u_n + 5 \\ u_0=100 \end{cases}$

On souhaite calculer la somme des N+1 premiers termes (u_0 à u_N). Pour ce faire, nous utiliserons l'algorithme suivant :

Variables
    I, N : entiers
    U, S : réels
Algorithme
    $U \leftarrow \ldots$
    $S \leftarrow \ldots$
    SAISIR N
    POUR I **ALLANT DE** 1 **À** N **FAIRE**
        $U \leftarrow \ldots$
        $S \leftarrow \ldots$
    FIN-POUR
    AFFICHER S
Fin Algorithme



1. compléter l'algorithme et l'organigramme correspondant
2. à l'aide d'un tableau faites tourner l'algorithme pour N=3
3. proposer une implémentation sous python

Dans de nombreuses situations, on aura besoin de simuler le hasard. Python propose le générateur pseudo-aléatoire de nombres entiers suivant :

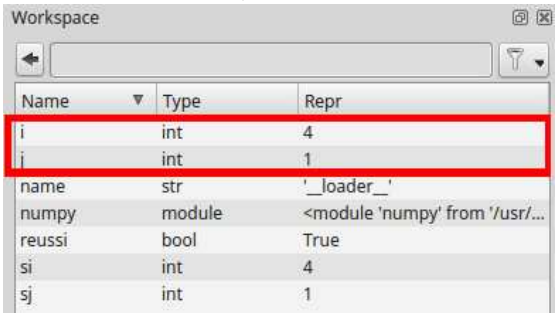
```
import numpy.random
numpy.random.randint(0,4)  # entre 0 et 4 exclu
```

`randint(n,m)` permet de tirer un nombre pseudo-aléatoire entier entre n et m-1.

Exo 3 : Vladimir cherche à implémenter une bataille navale contre l'ordinateur. Il considère que la mer est un carré de 5×5 et qu'un croiseur occupe une case au hasard.

```
import numpy.random
reussi=False
i=numpy.random.randint(0,5)
j=numpy.random.randint(0,5)
si=int(input())
sj=int(input())
if si==i and sj==j:
    reussi=True
if reussi :
    print("Bravo, croiseur coule !")
else :
    print("Le croiseur a survecu")
```

Dans le menu `Tools`, en sélectionnant `Workspace`. on peut trouver les valeurs des variables du programme (double-clique dessus pour mettre à jour)



1. proposer l'organigramme correspondant à ce programme,
2. l'implémenter sur python et le tester (vous pourrez utiliser le workspace pour pouvoir gagner facilement),
3. Vladimir aimerait avoir 5 tentatives pour trouver le croiseur ennemi. Il devra effectuer ses 5 tentatives puis seulement après il saura si le croiseur est coulé. Proposer une solution à l'aide d'une boucle POUR, tracer l'organigramme correspondant. Implémenter votre solution sous python.

Travail à faire pour la prochaine séance

Exo 4 :

On considère trois nombres réels x, y et z.

1. proposer un algorithme qui permettent d'ordonner ces 3 nombres (minimiser le nombre de comparaisons),
2. écrire cet algorithme en langage python.

Exo 5 :

On considère la suite (u_n) définie sur $\mathbb{N}$ par $\begin{cases} u_{n+1}=(n+1)u_n \\ u_0=1 \end{cases}$

1. calculer u_1, u_2, u_3 et u_4,
2. proposer un algorithme qui demande à l'utilisateur un entier n et renvoie u_n,
3. écrire l'organigramme correspondant.

rq : le terme u_n n'est autre que la factorielle de n (notée $n!$), ainsi $u_n = n! = \prod_{i=1}^{n} i = 1 \times 2 \times 3 \times \ldots n$.

Exo 6 :

On considère la suite de Fibonacci (u_n) définie par $u_0 = 0$, $u_1 = 1$ et $u_{n+1} = u_n + u_{n-1}$ pour $n \geqslant 1$.

1. calculer u_2, u_3, u_4 et u_5,
2. proposer un algorithme qui demande à l'utilisateur un entier n et renvoie u_n,
3. écrire (ou implémenter) cet algorithme en python.