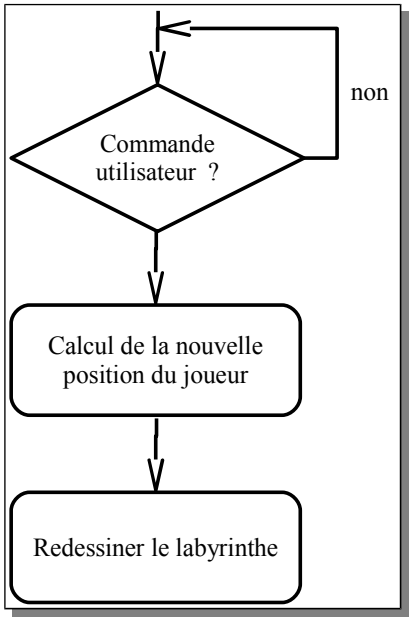
TD cours Python - 03 = Saisir des données

Le personnage (le "X") doit pouvoir se déplacer dans le labyrinthe. On doit donc être capable de lire le clavier pour saisir les commandes entrées par le joueur. Ces entrées devront être éventuellement converties puis traitées. Liste (non exhaustive) des notions du chapitre :

Fonctions : - Primordial pour organiser correctement un script.
Range : - Instruction de génération de liste.
Input : - Instruction pour lire le clavier.
Notes : - Pour générer une aide (ou un rappel) du rôle de la fonction.



1/ Les fonctions

Il est primordial de diviser un programme en petits bouts appelés **fonctions** en python. D'une part cela permet de **diviser la difficulté** et **tester chaque morceau indépendamment** mais aussi si la fonction réalisée est intéressante elle pourra être ajoutée à **une bibliothèque ou librairie personnelle et réutilisée avec de futurs programmes**.

Exemple de partage de notre programme en fonction :

Exemple de fonction :

Terminal
```
>>> def double(x) :
        return 2 * x

>>> print ("Le double de 3 est : " , double(3))
Le double de 3 est :  6
```

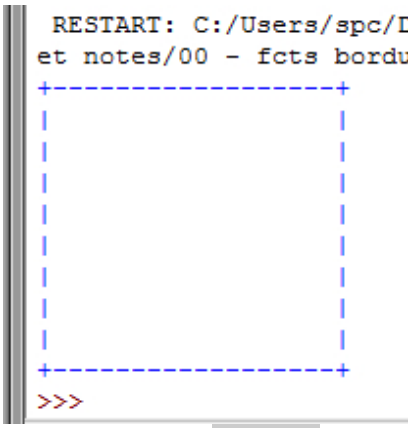
Dans ce cas, "print" appel la fonction "double" et la valeur 3 est transmise en **paramètre** à la fonction "double" et remplace alors "x". La fonction "double" **retourne alors le résultat** 6 qui remplacera la fonction "double(3) dans la fonction "print", cela revient à faire : print("Le double de 3 est : ", 6).

Donc une fonction (et c'est son rôle) retourne forcément une valeur. La fonction "Redessiner le labyrinthe" va dessiner le labyrinthe mais n'a pas de valeur à retourner. Dans ce cas python va donc ajouter et donc retourner la valeur **None**. C'est comme si on avait ajouté la ligne "return None" à la fin de cette fonction.

Exemple de fonction permettant l'affiche des bords d'un labyrinthe :

Script
```
def affiche_bordures(Nl,Nc) :
        print("+{}+" . format ("-" * (Nc  - 2)))
        for i in range (Nl - 2) :
                print("|{}|".format(" " * (Nc - 2)))
        print("+{}+" . format ("-" * (Nc  - 2)))

affiche_bordures(10,20)
```



Attention, en python il n'y a pas de caractère de fin de bloc. Le repérage d'un bloc s'effectue par **décalage**. Ce décalage s'appelle l'**indentation**. On dit **indenter** ou **désindenter** pour décaler à droite ou à gauche un texte.

2/ L'instruction range

Dans le script précédent une nouvelle instruction "**range**" est apparue. Elle permet de créer une liste, elle est donc utile avec la boucle "**for**".

Par exemple :

Terminal
```
>>> list (range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> debut = 1
>>> fin = 10
>>> list (range(fin))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list (range(debut,fin))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> saut = 2
>>> list (range(debut,fin,saut))
[1, 3, 5, 7, 9]
```

Dans cet exemple range est transformé en liste avec "**list**". Cela permet de voir comment on fabrique :

- Une liste avec un chiffre.
- Une liste qui part de 0 à la valeur d'une variable, attention "variable - 1".
- Une liste qui part de debut + 1 et qui se termine par fin - 1.
- Une liste avec un saut entre deux valeur (appelé **incrément** d'où le i dans les exemples) différentes de 1. On appelle **décrément** le saut entre deux valeurs qui diminuent.

3/ Saisir une entrée = input

Comme un programme est censé traiter des données, il faut pouvoir les entrer (**input**) et les sortir () où l'on a vu "print" qui est une sorte de sortie.

L'instruction "**input**" permet d'afficher une **invite** (prompt) et **retourne une chaîne de caractères** comme le montre l'exemple.

Terminal
```
>>> input ("Quelle direction ? ")
Quelle direction ? haut
'haut'
```

Évidemment on peut stocker cette entrée dans une variable. Mais si vous voulez rentrer un nombre il faudra convertir l'entrée.

Dans l'exemple ci-contre "val" est une chaîne de caractère et on n'a donc pas le droit de l'ajouter à une valeur numérique, ici 3.

Alors que ValNum est bien un entier que l'on peut ajouter sans problème.

Terminal
```
>>> val = input ("Entrer un nombre : ")
Entrer un nombre : 12
>>> 3 + val
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    3 + val
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> ValNum = int ( input ("Entrer un nombre : "))
Entrer un nombre : 12
>>> 3 + ValNum
15
```

4/ Des notes (commentaires) pour avoir de l'aide

Il est primordial de commenter vos lignes de programme. Déjà vous aurez une mauvaise note, ce qui est déjà une bonne raison, mais on peut vous promettre que **le lendemain vous serez incapable de vous rappelé pourquoi vous avez écrit une ligne de programme la veille**.

Il faut donc commenter votre programme … mais pas trop. S'il y a trop de commentaires, ils sont alors illisibles. Ce n'est pas la peine de dire que dans "print(score)" on affiche le score, en effet tout programmateur sait ce que veut dire print.

Script
```
val = input ("Entrer un nombre : ") # val est une chaîne
```

Commenter une ligne est une des bases de tout écriture de programme et cela quelque soit le langage. Mais le python offre une particularité intéressante :

En python on peut ajouter une aide dans une fonction. Par exemple on peut modifier ainsi la fonction de définition des bordures déjà écrite.

Script
```
def affiche_bordures(Nl,Nc) :
    """
        Affichage des bordures du labirynthe
        Nc = nombre de colonnes
        Nl = nombre de lignes
    """
    print("+{}+" . format ("-" * (Nc  - 2)))
    for i in range (Nl - 2) :
        print("|{}|".format(" " * (Nc - 2)))
    print("+{}+" . format ("-" * (Nc  - 2)))
```

Si le script à été lancé une fois, cette fonction est en mémoire (vive).

Lors du développement d'un programme, il y a des chances (des risques) de ne plus se rappeler ce que l'on a écrit dans une fonction particulière. Il suffit alors d'utiliser l'instruction "**help()**" dans la console comme le montre l'exemple suivant :

Terminal
```
>>> help (affiche_bordures)
Help on function affiche_bordures in module __main__:

affiche_bordures(Nl, Nc)
    Affichage des bordures du labyrinthe
    Nc = nombre de colonnes
    Nl = nombre de lignes
```

Vous devez donc absolument ajouter une aide à vos fonction, appelée **docstrings**.

Cette aide doit au minimum comporter une description succincte mais aussi est surtout la description de ses entrées et de ses sorties.