Algorithmique

et Python (2)

Plan

Définition

Une **fonction** est une suite d'instructions

- défini un sous-programme
- renvoie éventuellement un **résultat**

Une fonction peut avoir besoin de **paramètres** : des valeurs à entrer dans la fonction.
Elle peut renvoyer un **résultat**.

Exemple : print() et sqrt() sont des fonctions.

- print()
  - Nécessite un ou plusieurs arguments : print ("Hello"), print(a, b, c).
  - Affiche quelque chose à l'écran, mais ne renvoie pas de résultat
- sqrt()
  - nécessite un argument : sqrt(9)
  - renvoie un résultat sqrt(9) $\rightarrow$ 3.

Créer une fonction en Python

def *NomDeLaFonction* (parametre1, parametre2, ...) :

    instructions de la fonction
    avec un décalage vertical (indentation) obligatoire
    tout ce qui est décalé est dans la fonction
    return résultat

Remarque :

- return non obligatoire
  - Si il y est, la fonction s'arrête à cet endroit et renvoie le résultat
  - Si il n'y est pas, la fonction s'arrêtera à la fin des instructions et renvoie None
- NomDeLaFonction (p1, p2, ...) . p1, p2, ... = les arguments de la fonction.
  S'il n'y a pas d'arguments, on écrit NomDeLaFonction()

Utiliser une fonction en Python

- `NomDeLaFonction (valeur1, valeur2 ...)`
  Appèle la fonction en remplaçant les paramètres par les valeurs données et effectue les instructions.
- Pour récupérer le résultat, il faut le stocker dans une variable : *MaVariable* `= NomDeLaFonction (valeur1, valeur2 ...)`.

Exemple : Calculer le périmètre d'un carré de coté 3, de coté 4, de coté 5,6 et de coté 13,1.

Python

```
def  perimetreCarre(cote):
  perimetre=4*cote
  return perimetre

print("Pour 3, le perimetre = ", perimetreCarre(3))
print("Pour 4, le perimetre = ", perimetreCarre(4))
print("Pour 5.6, le perimetre = ", perimetreCarre(5.6))
print("Pour 13.1, le perimetre = ", perimetreCarre(13.1
```

Exemple : Définir la fonction $f(x) = 3x^2 + 2x - 4$ et récupérer dans une variable $f(-1)$ et $f \circ f(2)$.

Python

```
def f(x):
  return (3*x**2+2*x -4)
a=f(-1)
b= f(f(2))
```

print() $\neq$ return ()

- `print(Truc)` affiche Truc à l'écran, mais Truc n'est sauvegardé !
- `return(Truc)` ne se trouve qu'à l'intérieur d'une fonction. Truc est ce que renvoie la fonction quand elle est appelée.

Par exemple, `perimetreCarre(5)` peut être utilisé dans un calcul par la suite.

Exemple : Une fonction sans return

Python

```
 def Rien():
  a=2+4
  b=a*3
  print("il ne se passe rien")

Rien()
   # il va s'afficher "il ne se passe rien"
print(Rien())
  # il va s'afficher "il ne se passe rien"
  # puis None
```

Plan

Définition

Une liste (type list en Python) est une collection ordonnée d'objets que l'on groupe dans une seule variable. Les objets sont séparés par des virgules et l'ensemble est enfermé dans des crochets.

♡ =[objet0 , objet1 , objet 2 , ... , objetN]

On peut créer une liste vide en faisant ♡ =[ ]

Exemple : maListe=["a", "b", "c", 5, 8.3, "d"]

Extraire/modifier un élément d'une liste

♡ [n] donne l'élément numéro *n* de la liste ♡ (à partir de 0)
♡[0] le premier objet de la liste
♡[1] le deuxième objet ... ♡[-1] le dernier objet !

♡ [n] = ♠ met ♠ en position *n* dans la liste ♡ (en effaçant ce qui s'y trouvait).

Exemple : maListe=["a", "b", "c", 5, 8.3, "d"]

- maListe[1] → "b"
- maListe[6] → erreur !
- maListe[4]=7 donne maListe=["a", "b", "c", 5, 7, "d"

Extraire/modifier une partie d'une liste

♡[n :m] crée une liste contenant les objets du numéro *n* (inclus) au numéro *m* (exclus).

♡[n :m]= la liste des nouveaux éléments modifie les éléments du numéro *n* (inclus) au numéro *m* (exclus) de la liste.

Exemple : maListe=["a", "b", "c", 5, 8.3, "d"]
maListe[1 :3] → ["b", "c", 5].
maListe[1 :3]= [4, 'u', 0] modifie maListe=["a", 4, 'u', 0, 8.3, "d"]

Faire une liste à partir d'une chaine de caractère

list(♡) avec ♡ une chaine de caractère va créer une liste de tous les caractères de la chaine de caractère.

Exemple : list("bonjour) crée la liste ['b','o','n','j',o","u','r']

Opérations avec des liste

- tester si un objet ♠ appartient à une liste ♡

  `♠ in ♡`

  → True si l'objet est dans la liste

  → False sinon.

- `+` concaténer (coller) deux listes

Exemple : $[1, 2, 3] + [6, 5, 4]$ donne $[1, 2, 3, 6, 5, 4]$.

Python

```
ListeDeNombre=[2,3,6,9]
3 in ListeDeNombre        # donne True
7 in ListeDeNombre     # donne False
ListeDeNombre[3]=10    # donne [2,3,6,10]
```

Fonctions

ne modifient pas la liste passée en paramètre

- `len(♡)` donne le nombre d'élément de la liste ♡
- `min(♡)` donne le plus petit élément de la liste ♡
- `max(♡)` donne le plus grand élément de la liste ♡

Exemple :

Python

```
UneListe=[3,2,5,7,5,4]
len(UneListe)      # donne 6
min(UneListe)      # donne 2
max(UneListe)      # donne  7
```

Méthodes ça modifie la liste !

le nom de la liste . le nom de la méthode(paramètres supplémentaires)

- ♡.index(x) donne la première position de x dans la liste ♡ (en commençant à compter à 0)
- ♡.append(x) rajoute x à la fin de ♡
- ♡.insert(n,x) insère x en position n dans la liste ♡ (en décalant ce qui est après)
- ♡.remove(x) supprime la première apparition de x dans ♡
- ♠= ♡.pop(n) retire l'objet en position n de la liste ♡ et le stocke dans ♠.
- ♡.sort() trie les objets dans l'ordre croissant dans la liste ♡.
- ♡.reverse() inverse l'ordre des éléments dans la liste ♡

Exemple :

Python

```
ListeDeNombre=[2,3,6,9]
ListeDeNombre.index(3)    # donne  1
ListeDeNombre.append(7)
  # modifie [2,3,6,9,7]
a= ListeDeNombre.pop(2)
  # modifie [2,3,9,7] et donne a=6
```

Attention Une liste est modifiable. Il faut être vigilant si on modifie une liste en cours de programme. Si on veut une liste non modifiable, il faut utiliser un tuple.

Attention (bis) Dupliquer une liste.

Python

```
liste1= [1,2,3]
liste2=liste1
liste2[0]=7
```

On obtient liste2 = [7,2,3], mais aussi liste1 = [7,2,3] ! !
Si on veut dupliquer une liste et la modifier indépendamment de l'original, il faut utiliser des procédés spéciaux de copie : `liste2=list(liste1)` ou `liste2=liste1[ :]` (crée une liste extraite de liste1 en prenant tous les éléments).

Plan

Définition

Dans un algorithme, une boucle est une suite d'instructions qu'on répète en boucle un certain nombre de fois.

Définition

on dit que la boucle est (bornée) lorsqu'on sait exactement combien de fois la boucle va se répéter.

---

for *uneVariable* in *ListeDeValeur* :

> instructions à faire avec la variable *uneVariable*
> indentation obligatoire
> Toutes les valeurs de la *ListeDeValeur* sont prises l'une après l'autre

ListeDeValeur peut-être :

- une liste de nombre, ou de tout autre objet.
- un string : la variable va parcourir les caractères.
- `range(n)` la liste des entiers de 0 à $n-1$
- `range(k,n)` la liste des entiers de k à $n-1$.
- `range(k,n,p)` la liste des entiers de k à $n-1$ avec un pas de p (c'est à dire un entier sur p)

Exemple :

Python

```
for i in range(10) :
  print i
  # affiche la liste des entiers de 0 à 9
for i in ["a", "e", "r"]:
  print i
  # affiche successivement a, e et r
```

Plan

Définition

On dit qu'une boucle est non bornée lorsqu'on ne sait pas à l'avance le nombre d'itérations de la boucle nécessaire. Un test d'arrêt est effectué à chaque passage de la boucle : si la condition est True (vraie), on continue. Si la condition est False (faux), on s'arrête.

`while` *uneCondition* `:`

```
    instructions à faire tant que uneCondition est
    vraie avec indentation obligatoire
```

Danger ! le test d'arrêt est obligatoire et il faut s'assurer que la condition d'arrêt soit vérifiée à un moment. Sinon ça crée une boucle infinie qui fait planter le programme.

Itérations et boucle Une manière d'utiliser *while* est de l'utiliser avec un compteur qu'on augmente (ou diminue) à chaque tour de la boucle. On peut utiliser la valeur de ce compteur pour le test d'arrêt.

Exemple : Le code suivant affiche tous les entiers de 0 à 99

Python

```
n=0
while (n<=100) :
  print(n)
  n=n+1
```

Exemple : On cherche une valeur approchée à 10^{-2} de $x = \sqrt{13}$, c'est-à-dire x tel que $x^2 = 13$. On commence avec une valeur de $x = 3$ car $x^2 = 9$ est proche de 13 et on teste tous les nombres de 0,01 en 0,01 jusqu'à trouver celui dont le carré dépasse tout juste 13.

Python

```
x=3
while (x**2 <13) :
  x=x+0.01
print(x)     # résultat 3.609999999999987
```

Remarque : On ne peut pas mettre comme condition d'arrêt que $x^2 = 13$ car il n'y a aucune chance de tomber pile dessus !

Et il en reste quoi ?

1. Comment définir une fonction $f(x)$ en Python ?
2. Comment créé la liste contenant les éléments suivants : 1, 4, 6, 9 ?
3. Quel fonction sert à trouver la longueur d'une liste ?
4. Comment créer une boucle qui parcourt les entiers de 1 à 10 ?
5.

Python

```
x= 1
while (x<10) :
  x=x-1
```

Quel problème pose cette boucle ?

1. Comment définir une fonction $f(x)$ en Python ? def f(x) : puis les instructions avec indentation.
2. Comment créé la liste contenant les éléments suivants : 1, 4, 6, 9 ? [1, 4, 6, 9]
3. Quel fonction sert à trouver la longueur d'une liste ? len()
4. Comment créer une boucle qui parcourt les entiers de 1 à 10 ? for i in range(1,11) : instructions.
   ou alors i=1, while (i¡11) : i=i+1
5. Quel problème pose la boucle ? Elle ne s'arrête jamais.