Commandes python

Vues jusqu'ici

Opérations de base

Maths	+	-	×	÷	Quotient	Reste	Puissance
Python	+	-	*	/	//	%	**

abs(x) : donne la valeur absolue de la quantité stockée dans x
int(x) : converti x en entier / tronque x à l'unité
round(x) : arrondi x à l'unité **round(x,n)** : arrondi x à n chiffres derrière la virgule
int('….',b) : converti en base 10 le nombre entre guillemets originellement en base b.
bin(n) : converti en binaire le nombre « n »
hex(n) : converti en hexadecimal le nombre « n »
les nombres en base binaire s'ecrivent 0b…. par exemple $(101)_2$ s'écrira 0b101
les nombres en base hexadécimale s'écrivent 0x….. par exemple $(1B3)_{16}$ s'écrira 0x1B3

Librairie math **sqrt(x)** : donne la racine de x
pi : π cos(), sin() et tan() sont les fonctions trigo habituelles
degrees(x) : convertis en degré l'angle x initialement exprimé en radians.

Librairie random **randint(a,b)** : donne un entier aléatoire compris entre a (inclus) et b (exclu)
random() : donner un réel aléatoire entre 0 inclus et 1 exclu.

Typage :
type(vari) : donne le type de « vari » une variable.

Entrées sorties
print("chaine de caractère",variables) écris à la suite les expressions entre parenthèses et les contenus des variables. En rajoutant **end="€"** on remplace le retour à la ligne par le caractère €.
input("question") : récupère la chaine de caractères offerte par l'utilisateur en réponse à la question.

Chaines de caractères
"expression1"+"expression2" concatène les deux chaines et donne "expression1expression2"
len(chaine) : donne le nombre de caractères dans la chaine de caractère « chaine »

Structures conditionnelles
une condition est un test pouvant être vrai (True) ou faux (False)
elle est de type 'bool' c'est-à-dire BOOLéEN
comparateurs :
== (égal), != (différent), >= (supérieur ou égal), <=, >, <,
is (identique en valeur et en type),
in (appartenance à une ensemble),
opérateurs : and, or, not, parenthèses

```
if condition :
        instructions
elif condition alternative :
        instructions alternatives
else :
        instructions pour les cas
        non traités dans les
        conditions précédentes
suite du programme
```

boucles

- En général pour les boucles en **for** on utilisera un ensemble créé avec la fonction **range**. Celle-ci donne un éventail de valeurs, cette fonction peut utiliser plusieurs un ou plusieurs arguments

range(fin) : éventail des valeurs de 0 à fin-1
range(déb,fin) : éventail des valeurs de « déb » à fin-1
range(déb,fin, pas) on va avoir les valeurs de déb à fin-1 ,
en rajoutant « pas » à chaque itération.

```
for variable in ensemble :
        instructions
suite du programme
```

Durant l'exécution de la boucle la variable prendra successivement toutes les valeurs de l'ensemble.

On peut interrompre totalement l'exécution d'une boucle avec **break**
On peut interrompre une itération de la boucle avec **continue**, on passe alors à l'itération suivante.

- Pour les boucles en while, on devra utiliser une condition qui sera écrite sur le même modèle que pour les structures conditionnelles : expression A/comparateur/ expression B. Attention autant la variable évolue automatiquement dans les boucles for, autant pour celles en while, il faudra que dans la partie instructions la situation évolue.

```
while condition :
        instructions
suite du programme
```

Chaines de caractères
Si *chaine* est une variable aléatoire contenant une suite de caractères alors :

- pour accéder à l'élément de rang i on tape **chaine[i]**
- pour accéder à tous les éléments à partir du rang i on tape **chaine[i :]**
- pour accéder à tous les éléments jusqu'à celui de rang j on tapera **chaine[:j+1]**
- pour accéder à aux éléments dont le rang est entre i et j on tapera **chaine[i :j+1]**

comme pour range, la limite supérieure est toujours exclue
attention le premier élément a pour rang 0, il y a donc un décalage.
Pour concaténer (coller) chaine1 et chaine2 on tapera **chaine1+chaine2**
Pour accéder au code ascii d'un caractère car1 on tapera **ord(car1)**
Pour accéder au caractère donc le code ascii est code1 on tapera **chr(code1)**

Créer une fonction
Pour créer une fonction truc de paramètres par1, par2,… et ayant pour sorties s1,s2,…
Remarques :

```
def fonction(par1,par2,…) :
        instructions
        return s1,s2,…
```

- une fonction peut ne pas avoir de paramètres, elle peut aussi ne pas avoir de sortie
- Si une fonction n'est pas appelée, elle restera à l'état de potentiel et ne s'activera pas.
- Les variables créées lors de l'exécutions d'une fonction, seront automatiquement détruite à la fin de l'exécution de celle-ci.
- Quand on appelle une fonction les paramètres seront remplacés par des valeurs qui seront alors appelées arguments.

Gestion des erreurs
Quand on anticipe qu'une commande comme une assignation peut provoquer une erreur, on demandera à python d'essayer de l'exécuter, et on lui offrira un plan B si la tentative pose un problème, suivant la nature de celui-ci.
Il existe un grand nombre d'erreurs potentielles, parmi celle-ci nous avons : ValueError, TypeError, NameError (variable non définie), ZeroDivisionError

```
try :
        instruction
except nom de l'erreur 1 :
        instructions à exécuter
        en cas d'erreur 1
except nom de l'erreur 2 :
        instructions à exécuter
        en cas d'erreur 2
suite du programme
```

Listes
Elles se gèrent d'une manière très similaire aux chaines de caractères
(en tout cas pour les accès et pour les concaténations), contrairement à ces dernières on peut changer un élément.
Pour rajouter un élément elt à une liste liste1, on utilise la commande append : **liste1.append(elt)**
Création d'une liste :
Reproduction d'un motif « listebase » à l'identique n fois : liste=listebase*n
Exemple : motVide=[''_'']*len(motmystère)
Création par compréhension (avec une formule)

Liste = [formule for i in listePertinente]
Exemple : si on veut stocker les 10 premiers termes de la suite (u_n) définie par $u_n = 3n - 4$
Liste=[3*n-4 for i in range(10)]