Formulaire de syntaxe Python

LINGE1225 – Programmation en Economie et Gestion – 2020-2021

Types de variables et affectations

boolean = `True/False`
integer = `5`
float = `3.14`
string = `"123abc"` ou `'abc123'`
list = `[v1, v2, ...]`
dictionnaire = `{clé1:v1, clé2:v2, ...}`

Affectation multiple:
```
x, msg = 42,"bonjour"
#Commentaires marqués par un "#"
```

Opérateurs numériques

+	addition
−	soustraction
*	multiplication
/	division
//	division entière
%	modulo
**	exposant

Opérateurs logiques

==	égal
!=	différent
>	supérieur
<	inférieur
>=	supérieur ou égal
<=	inférieur ou égal
and	**et** logique
or	**ou** logique
not	**non** logique

Affectation abrégée

Opérateur	Exemple	Forme équivalente
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Liste des mots réservés

and	as	assert	break
class	continue	def	del
elif	else	except	False
finally	for	from	global
if	import	in	is
lambda	None	nonlocal	not
or	pass	raise	return
True	try	while	with
yield			

Fonctions pré-implémentées

Fonctions générales:

str(x)	convertis x en **string**
int(x)	convertis x en **integer**
float(x)	convertis x en **float**
type(x)	renvoie le type de x
abs(x)	renvoie la valeur absolue de x
round(x,n)	arrondis x à n décimales
len(x)	renvoie la longueur de x
range(x,y,z)	renvoie la liste des nombres dans [x,y[par pas de z

Fonctions et constantes du Module Math:

sin(x)	le sinus de x (radians)
cos(x)	le cosinus de x (radians)
tan(x)	la tangeante de x (radians)
log(x)	le logarithme népérien de x
log(x,y)	le logarithme de x en base y
sqrt(x)	la racine carrée de x
pi	la constante π
e	la constante e

Syntaxe de base

Importation d'un module complet:
```
import <module>
```

Importation d'une fonction spécifique:
```
from <module> import <fonction>
```

Définition d'une fonction personnalisée:
```
def <fonction>(<paramètres>):
    <code>
    return <valeur>
```

Conditionnelles et boucles

Syntaxe d'une instruction conditionnelle:
```
if <condition>:
    <code>
elif <condition>:
    <code>
...
else:
    <code>
```

Syntaxe des différents types de boucles:
```
while <condition>:
    <code>

for <indice> in range(start,stop,step):
    <code>

for <variable> in <liste>:
    <code>
```

Contrôles relatifs aux boucles:

break	arrête la boucle en cours
continue	passe à l'itération suivante

Chaînes de caractères

Concaténation (+) et répétition (*):
`"abc"+"def"` → `"abcdef"`
`"abc"*3` → `"abcabcabc"`

Soit `s` une chaine de caractère.

Opérateurs disponibles:

s[i]	renvoie le caractère en position i
s[i:j]	renvoie les caractères situés dans l'intervalle [i,j[

Si `i` est négatif, `s[i]` renvoie le i^e caractère en partant de la fin (e.g. `"abc"[-1]`→`"c"`)

Fonctions disponibles:

s.upper()	converti s en majuscules
s.lower()	converti s en minuscules
s.split(x)	fractionne s en utilisant x comme séparateur
s.find(x)	renvoie l'indice de x dans s
s.count(x)	compte le nombre d'occurences de x dans s

Opérateurs sur les listes

Soit une liste `L`. Opérateurs disponibles:

L=[]	création d'une liste vide
L[i]	renvoie l'élément à l'indice i
L[i:j]	renvoie les éléments situés dans l'intervalle [i,j[
L[i] = x	place x dans L à l'indice i
del L[i]	supprime l'élément en position i

Si `i` est négatif, `L[i]` renvoie le i^e élément en partant de la fin

Fonctions sur les listes

Soient `L` et `L2` deux listes.

Fonctions disponibles:

L.append(x)	ajoute x à la fin de L
L.extend(L2)	ajoute les éléments de L2 à la fin de L
L.pop(i)	supprime l'élément d'indice i et renvoie sa valeur
L.clear()	supprime les éléments de L
L.copy()	renvoie une copie de L

Dictionnaires

Déclaration d'un dictionnaire vide: `d={}`

Opérations sur les dictionnaires:

d[k]	renvoie l'élément associé à la clé k (= un string)
d[k] = v	associe la valeur v à la clé k
del d[k]	supprime l'élément associé à k
len(d)	renvoie le nombre d'éléments de d

Test d'appartenance:
```
if <clé> in <dict>:
    <code>
```

Les boucles sur les dictionnaires:
```
for <clé> in <dict>:
    <code>

for <clé>,<valeurs> in <dict>.items():
    <code>
```

Objets et héritage

Syntaxe générale pour une classe:
```
class <classe>:
    <attributs de classe>
    <constructeur et méthodes>
```

Méthodes spécifiques aux classes:
- Constructeur:
```
def __init__(self,<attributs>):
    <code>
```
- Impression:
```
def __str__(self):
    <code>
```

Accès à un attribut `x` via `self.x`

Syntaxe pour une sous-classe:
```
class <classe>(<super-classe>):
    <constructeur et méthodes>
```

Constructeur d'une sous-classe:
```
def __init__(self,<attributs>):
    <super-classe>.__init__(self,...)
    <code>
```

S'il y a plusieurs super-classes (héritage multiple), la synthaxe devient:
```
class <classe>(<sc1>,<sc2>,...)
```

Fonctions utiles sur les objets:

isinstance(x,y)	renvoie True si l'objet x est une instance de la classe y et False sinon
issubclass(x,y)	renvoie True si la classe x est une sous-classe de la classe y et False sinon

Attention : Ce formulaire contient la liste des fonctions que vous êtes autorisés à utiliser durant l'examen. Sauf mention contraire explicite à l'examen, l'utilisation de fonctions non-listées dans ce formulaire entraîne une pénalité.