10/10/2013

Mémento des commandes basiques en Python 3.0

Raccourci vers la programmation

Mémento des commandes basiques en Python 3.0

Raccourci vers la programmation

Généralités

Non déclaration des variables

Typage automatique des variables

Mots réservés:

and	assert	break	class	continue	def
del	elif	else	except	exec	finally
for	from	global	if	import	in
is	lambda	not	or	pass	print
raise	return	try	while	yield	

Affectation	=	`n = 5`	`⇒ 5`
Affectation multiple	= =	`x = y = 7` `a, b = 1, 3`	`⇒ 7 7` `⇒ 1 3`

Opérateurs	+ - * / // %	`1*2+8/5`
		`10 / 3`
		`10 // 3`
		`10 % 3`
			`⇒ 3.6`
			`⇒ 3.3333333335`
			`⇒ 3`
			`⇒ 1`

Opérateurs de comparaison	== != > < >= <=		
Instruction sur plusieurs lignes	\	`ch = "azerty\` `qsdfgh"`	`⇒ azertyqsdfgh`
Commentaire	#	`# ceci est un commentaire`	
Insertion de codes spéciaux	\	`print('coucou l\'ours')`	`⇒ coucou l'ours`
triples quotes	"""	`al = """bonjour` `{l'oiseau#"""`	`⇒ bonjour` `{l'oiseau#`

Blocs d'instruction:

```
Ligne d'en-tête:
    première instruction du bloc
    ... ...
    dernière instruction du bloc
```

Entrées-Sorties

Input, print

```
prenom = input('Entrez votre prenom : ')
print('Bonjour,', prenom)
nn = int(input('Veuillez entrer un
nombre positif quelconque : ',))
print('Le carre de', nn, 'vaut', nn**2)
```

```
⇒ Entrez votre prenom : toto
⇒ Bonjour, toto
⇒ Veuillez entrer un nombre
  positif quelconque : 4
⇒ Le carre de 4 vaut 16
```

Importation de modules de fonctions

```
from module import *              # importe toutes les fonctions du module
from module import nomfct        # importe nomfct du module
import module                    # importe tout le module (fct + objets)
     # dans ce cas les fonctions sont appelées par la syntaxe
module.fonction()
```

Fichiers

Positionnement dans le répertoire de travail:

```
from os import chdir, getcwd      # importe chdir et getcwd du module os
# attention! une fonction open existe dans os qu'il ne faut PAS importer
chdir("/home/jules/exercices")    # changement de répertoire
getcwd()                          # affichage du répertoire courant
```

Lecture / Écriture

Ouverture	obFichier = open(nomFichier, mode) # monFichier est le nom du fichier à ouvrir #mode: • a : ajout, écrit à la fin du fichier • w : write, crée/écrase et écrit • r : lecture seule	`ofi = open(monFichier,'a')`	

Écriture	obFichier.write('texte à écrire')	`obfi.write('Bonjour, fichier! ')`
		`obfi.write("Quel beau temps, aujourd'hui!")`
		`obfi.write(str(124))`	

Lecture d' 1 ligne: tout:	obFichier.read() obFichier.read(*nbCaracteresALire*) obFichier.readline( ) obFichier.readlines( )	`print(obfi.read())` renvoie 1 chaine renvoie une liste de chaines (ue chaine par ligne)	`⇒ Bonjour, fichier! Quel beau temps, aujourd'hui! 124`
Fermeture	obFichier.close()	`obfi.close()`	⇒

Instructions conditionnelles

if … elif …else …

```
a = int(input('Entrez la valeur de a à tester'))
if a > 0 :
     print("a est positif")
elif a < 0 :
     print("a est négatif")
else:
     print("a est nul")
```

Itérations

for … in …

```
for element in sequence:
    bloc d'instruction
```

while

```
while (condition d'arrêt):
    indentation du compteur
    instructions
```

Break : pour sortir d'une itération :

```
while <condition 1> :
    --- instructions diverses ---
    if <condition 2> :
        break
```

Erreurs

Erreurs de syntaxe

Erreurs sémantiques = erreur logique (liée à l'algorithme)

Erreurs à l'exécution = exceptions

Gestions des exceptions: try-except-else

```
def existe(fname):
    try:
        f = open(fname,'r')
        f.close()
        return 1
    except:
        return 0

filename = input("Veuillez entrer le nom du fichier : ")
if existe(filename):
    print ("Ce fichier existe bel et bien.")
else:
    print ("Le fichier", filename, "est introuvable.")
```

Fonctions

Définition:

```
def nomDeLaFonction(liste de paramètres):
    ...
    bloc d'instructions
    ...
    return valeur
```

Utilisation:

```
n=nomDeLaFonction(paramètres)
```

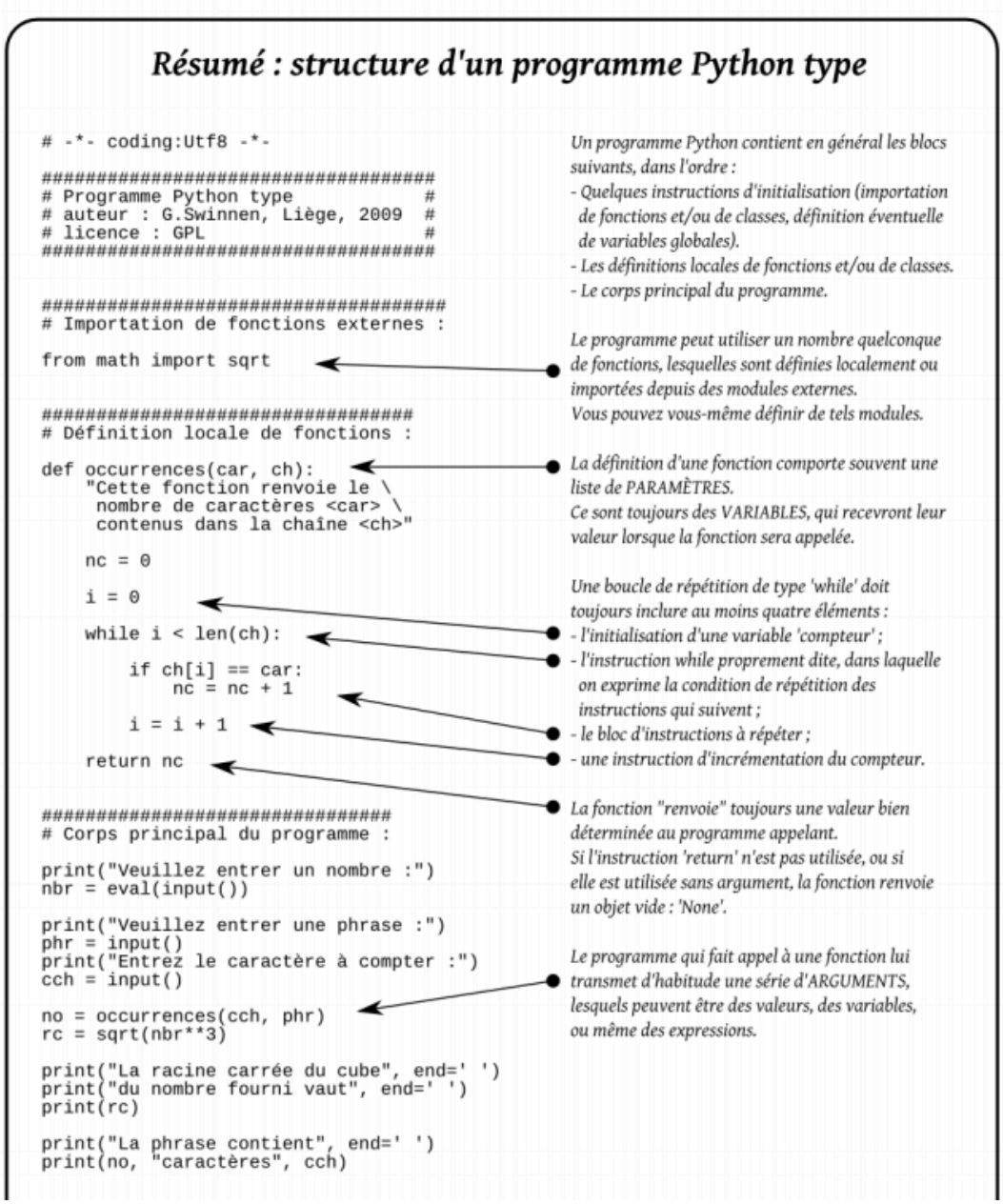

Figure 1 : Structure d'un programme Python Type - http://inforef.be/swi/download/apprendre_python3_5.pdf

Structures de données

Séquences

On accède à un élément par son **index** (suite d'éléments ordonnés).

Chaînes de caractères '....'

Séquences **non modifiables**

Concaténation	+	`n= 'abc' + 'def'`	⇒ abcdef
Répétition	*	`m = 'zut' * 4`	⇒ zutzutzutzut
Indiçage	chaîne[indice]	`nom = "Cédric"` `nom[1]`	⇒ é
Conversion	int(chaine) float(chaine)	`ch = '8647'` `print(int(ch) + 45)`	⇒ 8692
Slicing	chaîne[début: fin+1]	`ch = "Juliette"` `print(ch[0:3])`	⇒ Jul
Longueur	**len**(chaine)	`len("Juliette")`	⇒ 8
Itération	**for** élément **in** séquence	`for c in "Jules":` `  print(c, end="")`	⇒ Jules
		`for c in ['er','re',2]:` `  print(c, end="")`	⇒ er re 2

Appartenance	**in** séquence	`voy, n = 'aeiouy', 'a'`
		`if n in voy:`
		`  print("OK")`
			⇒ OK

Les chaînes sont comparables (>, <, ==, >=, ...)

Méthodes classiques des chaînes de caractères:

- split() : convertit une chaîne en une liste de sous-chaînes. On peut choisir le caractère séparateur en le fournissant comme argument, sinon c'est un espace, par défaut
- join(liste) : rassemble une liste de chaînes en une seule. Attention : la chaîne à laquelle on applique cette méthode est celle qui servira de séparateur (un ou plusieurs caractères); l'argument transmis est la liste des chaînes à rassembler
- find(sch) : cherche la position d'une sous-chaîne sch dans la chaîne
- count(sch) : compte le nombre de sous-chaînes sch dans la chaîne
- lower() : convertit une chaîne en minuscules
- upper() : convertit une chaîne en majuscules
- capitalize() : convertit en majuscule la première lettre d'une chaîne
- swapcase() : convertit toutes les majuscules en minuscules et vice-versa
- strip() : enlève les espaces éventuels au début et à la fin de la chaîne
- replace(c1, c2) : remplace tous les caractères c1 par des caractères c2 dans la chaîne
- index(c) : retrouve l'index de la première occurrence du caractère c dans la chaîne

Les listes [...,...]

Séquences **modifiables**

Déclaration	liste1 = **[**elt1, elt2, elt3**]**		
Accès aux éléments	elt1 = liste1[0]		

Création	list(range(nb_element))
	list(range(1er nb, der_exclus, step))
		`range(3)`
		`range(5,7)`
		`range(5, 9, 2)`
			⇒`[0,1,2]`
			⇒`[5,6]`
			⇒`[5,7]`

Slicing avancé			
Insertion	liste[i: i]= [i]	`l = [1,2,3,5]` `l[3: 3]=[4]`	⇒`[1,2,3, 4,5]`
Suppression	liste[i: j]=[]	`l[:1]=[]`	⇒`[3,4,5]`
Remplacement	liste[i, j]=[elt4, elt5]	`l[:]=[6,7,8]`	⇒`[6,7,8]`

Parcours	**for** index **in range(len**(ch)):
	instructions
		`f = ['M','C']`
		`for index in range(len(f)):`
		`  print(index, f[index])`
			⇒`0 M`
			⇒`1 C`

Concaténation	+		
Répétition	*		
Appartenance	**in** séquence		

Méthodes objets applicables aux listes:

- sort() : tri
- append() : ajout d'un élément à la fin
- reverse() : inversion de l'ordre des éléments
- index(element) : retrouve l'index d'un élément
- remove(element) : effacement d'un élément

Fonction :

- del liste[index] : efface l'élément correspondant à l'index

Nombres aléatoires :

```
from random import *
```

- random() : renvoie un nombre réel aléatoire, de valeur comprise entre zéro et un
- randrange(p1,p2,step) : renvoie un entier compris entre p1 (=0 par défaut) et la valeur (p2-1). step=1 par défaut, sinon il indique le pas d'intervalle entre les entiers.

Les tuples (...,...)

Listes non modifiables. Intérêts : garanti la non modification des données , moins de place en mémoire.

Déclaration	tuple1 = **(**elt1, elt2, elt3**)**		
Accès aux éléments	elt1 = tuple1[index]		

Les dictionnaires {...,...}

Suite d'éléments non ordonnés, accessibles par une clé.

Déclaration	dico1 = {}	`dico = {}`	
Ajout d'éléments	dico1[cle1] = valeur1	`dico['computer'] = 'ordinateur'`	⇒
Accès à un élément	dico1[cle1]	`print(dico['computer'])`	⇒ ordinateur
Suppression d'élément	del dico1[cle]		
Accès aux valeurs non répertoriées	dico1.get(index, valeur_a_fournir_si_rien_a_l_index)		

Méthodes propres aux dictionnaires:

- keys() renvoie un itérateur avec la liste des clés utilisées dans le dictionnaire
- values() renvoie un itérateur avec la liste des valeurs mémorisées dans le dictionnaire
- has_key(clé) permet de savoir si un dictionnaire comprend une clé déterminée.
- items() renvoie un itérateur avec une liste de tuples (cle, valeur)
- copy() permet d'effectuer une vraie copie d'un dictionnaire.

Les objets

Déclaration

```
class NomObjet:
     "Description de l'objet pour générer la doc"
    def _init_(self, at1=val_defaut1, at2=val_defaut2):   #constructeur
         self.attribut1 = at1            #initiation des attributs
         self.attribut2 = at2

    def methode1(self):                  #méthode 1
         …
```

Héritage

```
class Fille(Mere):
   …
```

Résumé : définition et utilisation d'une classe

```
###################################
# Programme Python type           #
# auteur : G.Swinnen, Liège, 2009 #
# licence : GPL                   #
###################################

class Point(object):
    """point géométrique"""
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Rectangle(object):
    """rectangle"""
    def __init__(self, ang, lar, hau):
        self.ang = ang
        self.lar = lar
        self.hau = hau

    def trouveCentre(self):
        xc = self.ang.x + self.lar /2
        yc = self.ang.y + self.hau /2
        return Point(xc, yc)

class Carre(Rectangle):
    """carré = rectangle particulier"""
    def __init__(self, coin, cote):
        Rectangle.__init__(self,
                coin, cote, cote)
        self.cote = cote

    def surface(self):
        return self.cote**2

############################
## Programme principal : ##

# coord. de 2 coins sup. gauches :
csgR = Point(40,30)
csgC = Point(10,25)

# "boîtes" rectangulaire et carrée :
boiteR = Rectangle(csgR, 100, 50)
boiteC = Carre(csgC, 40)

# Coordonnées du centre pour chacune :
cR = boiteR.trouveCentre()
cC = boiteC.trouveCentre()

print("centre du rect. :", cR.x, cR.y)
print("centre du carré :", cC.x, cC.y)

print("surf. du carré :", end=' ')
print(boiteC.surface())
```

La classe est un moule servant à produire des objets. Chacun d'eux sera une instance de la classe considérée.

Les instances de la classe Point() seront des objets très simples qui possèderont seulement un attribut 'x' et un attribut 'y' ; ils ne seront dotés d'aucune méthode.

Le paramètre SELF désigne toutes les instances qui seront produites à partir de cette classe.

Les instances de la classe Rectangle() posséderont trois attributs. Le premier ('ang') doit être lui-même un objet de classe Point(). Il servira à mémoriser les coordonnées de l'angle supérieur gauche du rectangle. Les deux autres contiendront sa largeur et sa hauteur.

La classe Rectangle() comporte une méthode, qui renverra un objet de classe Point() au programme appelant.

Carre() est une classe dérivée, qui hérite les attributs et méthodes de la classe Rectangle(). Son constructeur doit faire appel au constructeur de la classe parente, en lui transmettant la référence de l'instance en cours de création (self) comme premier argument.

La classe Carre() comporte une méthode de plus que sa classe parente.

Pour créer (ou instancier) un objet, il suffit d'appeler une classe comme on appelle une fonction. La valeur renvoyée est une nouvelle instance de cette classe. Les instructions ci-contre créent donc deux objets de la classe Point() ...

... et celles-ci, encore deux autres objets.

Note : par convention, on donne aux classes des noms commençant par une majuscule.

La méthode trouveCentre() fonctionne pour les objets des deux types, puisque la classe Carre() a hérité de la classe Rectangle().

La méthode surface(), par contre, ne peut être invoquée que pour les objets Carre().

Figure 2 : Définition et utilisation d'une classe - http://inforef.be/swi/download/apprendre_python3_5.pdf

Modules

Fichier .py contenant des définition de classes.

Importation :

- **import** NomModule
- **from** NomModule **import** NomClasse

Lancement :

```
if __name__ == "__main__":
    …
```

Les instructions suivant le main ne seront exécutées que si le module est lancé en tant que programme.

Style Guide pour le code Python

http://www.biologeek.com/bonnes-pratiques,conferences,django,python,traduction/bonnes-pratiques-et-astuces-python/