Memento rapide des principales instructions en Python 3

Commentaires

`# ceci est un commentaire` `x=a+1 # commentaire après instruction`	Commence toujours par #. S'écrit sur une ligne ou à la suite d'une instruction

Types de base

		Exemples
Entier	`integer`	`45` `0` `-380`
Flottant ou décimal	`float`	`-45.23` `0.` `-380.` `2.0`
Complexe	`complex`	`2+1j` `2.1+0j` `-3.1j`
Booléen	`boolean`	`True` `False`
Chaîne de caractères	`string`	`"Thomas"` `'3 pommes ?'` `''` `"L'eau"` ou `'L\'eau'`

chaîne vide — apostrophe

Conversion entre types

Conversion en entier	`int(3.0)` `int('3')`	`3`
Troncature	`int(3.8)`	`3`
Conversion en décimal	`float(3)` `float('3.8')`	`3.0` `3.8`
Conversion en chaîne de caractères	`str(3)` `str(3.8)`	`'3'` `'3.8'`
Mise en forme d'une chaîne de caractères en majuscules ou minuscules	`'Maths'.upper()` `'Maths'.lower()`	`'MATHS'` `'maths'`

Variables

Le nom d'une variable commence toujours par une lettre. Il peut contenir des lettres en majuscule ou minuscule, des chiffres et le tiret bas.

`x` `abc` `Ma_var` `Var1`

Affectation d'une valeur à une variable :

directe

```
x=12+8
x=3.
Mot='math'
```

en fonction de la valeur d'une autre variable

```
x=2*a+1
Nom_complet=Prenom+' '+Nom
```

en fonction de sa propre valeur

```
x=2*x+1
Mot=Mot+'s'
```

Demande de saisie de valeur

```
phrase=input('entrer une phrase'))
x=int(input('entrer un entier'))
x=float(input('entrer un décimal'))
x=complex(input('entrer un complexe'))
```

Affectations multiples

```
a=b=c=5
a,b=0,1
a,b=b,a
```

Affectations simultanées : a←0 et b←1
Inversion des valeurs entre deux variables

Affichage de valeurs dans la console

Affichage simple

```
print(78)
print('3 pommes ? ')
print(x)
print(2*x+1)
```

Affichages multiples

```
print(a,b)
print(a+1,b**2-4)
print('la valeur de x est', x)
```

Opérations de base sur les variables et les valeurs

Nombres	
`+, -, *, /`	Opérations standards
`a//b, a%b`	Quotient et reste de la division euclidienne de `a` par `b`
`a**b, pow(a,b)`	Puissance a^b
`abs(x)`	Valeur absolue

`round(x,n)`
`round(3.59,1)`
`round(4.785)`
	Arrondi de `x` à `n` décimales.
	`3.6`
	`5`

Chaînes de caractères	

`+`
`'lycée'+'queneau'`
`'lycée'+'`
`'+'queneau'`
	Concaténation
	`'lycéequeneau'`
	`'lycée queneau'`

`len('maths')` `len(mot)`	`5` Longueur de la variable `mot`
`mot.upper()` `mot.lower()`	`mot` écrit en majuscules `mot` écrit en minuscules

Blocs d'instruction

Les instructions sont écrites en blocs indentés d'une tabulation sous la première ligne.

Instructions conditionnelles

```
if <test> :
      bloc d'instructions
```

```
if <test> :
      bloc d'instructions 1
else :
      bloc d'instructions 2
```

```
if <test 1> :
      bloc d'instructions 1
elif <test 2> :
      bloc d'instructions 2
…
elif <test n> :
      bloc d'instructions n
else : #facultatif
      bloc d'instructions
```

Boucles

Boucle non bornée

```
while <test> :
      bloc d'instructions
```

Boucle bornée

```
for <var> in <séquence>:
      bloc d'instructions
```

Tests numériques ou alphanumériques

Ordre

`<, <=, >, >=`

Égalité ou inégalité

`==, !=`

Opérateurs logiques

`<test1> and <test2>`

`<test1> or <test2>`

`not <test>`

Parcours d'une séquence

	`x` prend successivement …
`x in range(a)`	… les entiers de `0` à `a-1`
`x in range(a,b)`	… les entiers de `a` à `b-1`
`x in range(a,b,c)`	… les entiers de `a` à `b-1` avec un pas de `c` (nombre entier).
`x in 'thomas'`	… `'t'`, puis `'h'`, puis `'o'`, etc.…
`x in [4.1 , 7 , -89]`	… les valeurs `4.1`, puis `7` puis `-89` de la liste.

Parcours simultané de deux listes :

`x,y in zip([1,2],[a,b])` `(x,y)` vaut `(1,a)`, puis `(2,b)`.

Fonctions

Les instructions sont écrites en blocs indentés d'une tabulation sous la première ligne.

Définition d'une fonction

```
def ma_fonct (arg1, arg2, …) :
      bloc d'instructions
      return(valeur)
```

Utilisation de la fonction

```
x=ma_fonct(val1, val2, …)
```

Listes

`['antoine',3,-5.2]`	une liste s'écrit entre crochets et peut contenir divers éléments
`[k for k in range(5)]`	construit la liste `[0,1,2,3,4]`
`[k**2 +1 for k in range(9)]`	construit la liste $[u_0, u_1, u_2, u_3, \dots u_8]$ pour la suite $u_n = n^2 + 1$
`[3+k*0.1 for k in range(11)]`	construit la liste `[3,3.1,3.2,3.3,…,4]`

Memento des modules Python qui peuvent être utiles au lycée

Importation de modules

Il est préférable que cette instruction soit écrite en début de progamme.

`from nom_du_module import *`	importation de toutes les données du module
`from mon_du_module import elt1, elt2, …`	importation uniquement des éléments cités du module

Module de calcul dans $\mathbb{R}$: `math`

`pi, e`	valeurs approchées de π et e	`exp(x)` ou `e**x`	e^x
`floor(x)`	le plus grand entier plus petit ou égal à x	`log(x)`	$\ln(x)$
`trunc(x)`	le plus grand entier inférieur ou égal à x	`log10(x)`	$\log(x)$
`ceil(x)`	le plus petit entier supérieur ou égal à x	`sqrt(x)`	$\sqrt{x}$

Fonctions trigonométriques (valeurs de l'angle en radians)

`cos(x), sin(x), tan(x)` `acos(x), asin(x), atan(x)`

Conversions d'angles

`degrees(x)` `radians(x)`

Module de calcul dans $\mathbb{C}$: `cmath`

`1j` `3-2.7j`	i $3-2.7i$
`complex(rel,im)`	retourne le nombre complexe de parties réelle `rel` et imaginaire `im`
`z.real, z.imag`	parties réelles et imaginaires du complexe z
`abs(z)`	module $\lvert z\rvert$
`phase(z)`	argument de `z` compris dans $]-\pi;\pi]$
`polar(z)`	retourne le couple module, argument du complexe `z`
`z.conjugate()`	retourne le conjugué de `z`
`exp(x)` ou `e**x, log(x), log10(x), sqrt(x), cos(x), sin(x), tan(x), acos(x), asin(x), atan(x), e, pi`	comme pour le module `maths`, mais avec des valeurs complexes.

Module de génération de valeurs aléatoires : `random`

Nombre aléatoire	`randint(a,b)`	entier dans $[a;b]$ (a et b entiers)
	`random()`	décimal dans $[0;1[$
	`uniform(a,b)`	décimal dans $[a;b]$
`choice(liste)`	`choice(chaine)`	un élément au hasard de la `liste` ou de la `chaîne` de caractères

Module d'affichage graphique : `pylab`

Dans ce module, les listes sont présentées sous forme de tableau à une dimension : `[1,2,3]` sera sera affiché `[1 2 3]` ou `array([1,2,3])`.

Pour que le graphique s'affiche à l'écran, il faut mettre en dernière ligne du programme : `show()`

`arange(a,b,p)`	retourne la liste des nombres allant de `a` à `b` avec un pas `p`
`linspace(a,b,n)`	retourne la liste de n valeurs régulièrement réparties sur `[a ;b]`
`arange(-2,2.5,0.5)`	`[-2.0, -1.5, -1., -0.5, 0., 0.5, 1., 1.5, 2.]`
`linspace(2,3,6)`	`[2., 2.2, 2.4, 2.6, 2.8, 3.]`

`plot(listeX, listeY)`	trace une **ligne brisée** à partir des points dont les abscisses sont dans `listeX` et les ordonnées dans `listeY`	*
`scatter(listeX, listeY)`	trace un **nuage de points** dont les abscisses sont dans `listeX` et les ordonnées dans `listeY`	*
`bar(listeX,listeH)`	trace un **diagramme en barre**, `listeX` contient les abscisses des coins inférieurs gauches des barres, `listeH` les hauteurs.	* **
`boxplot(L)`	construit (sur la droite $x=1$) la **boite à moustache** verticale correspondant à la liste `L` passée en argument.	
`boxplot([L1,L2,…, Ln])`	construit n **boîtes à moustaches** correspondant aux listes `L1, … Ln`.	

`title('mon titre')`	ajoute un **titre** au graphique
`legend()`, `legend('best')`, `legend('right')`, `legend('left')`, `legend('center')`	permet d'afficher une **légende** au graphique en précisant (ou pas) sa position *(possibilité de rajouter `upper` ou `lower` avant ce mot de position. Ex : `'upper right'`).* Attention : il faut préalablement donner un nom à chaque courbe tracée : on rajoute l'argument `label='nom de ma fonction'` à la liste des arguments des instructions `plot`, `scatter`, `bar` ou `boxplot` utilisées.
`xlabel('abscisses')` `ylabel('ordonnées')`	place les **étiquettes** `abscisses` et `ordonnées` sur les axes respectifs.
`axis([xm,xM,ym,yM])`	définit la **fenêtre d'affichage** : l'axe des abscisses va de `xm` à `xM` et l'axe des ordonnées de `ym` à `yM`
`xlim(a,b)` `ylim(a,b)`	l'axe des abscisses sera limité à $[a;b]$ même principe pour l'axe des ordonnées
`xticks(listeX)` `yticks(listeY)`	**graduation** de l'axe des abscisses : `listeX` est la liste des valeurs affichées même principe pour l'axe des ordonnées
`grid()`	affiche une **grille** (adaptée à la graduation des axes)
`axhline()` `axvline()`	affiche respectivement les axes des abscisses et des ordonnées *

* On peut rajouter l'argument `color=…` pour la couleur des points. Les noms des couleurs sont à écrire en anglais : `'blue'`, `'red'`, …

** On peut rajouter l'argument `width=…` pour la largeur des barres.

Module de dessin : `turtle`

Pour que le tracé apparaisse à l'écran, il est nécessaire d'écrire en dernière ligne du programme : `mainloop()`
Par défaut, le crayon est placé au centre de la fenêtre, ses coordonnées sont (0,0), la direction du tracé est vers la droite.

`down()` `up()`	permet de baisser ou de lever le crayon afin d'écrire ou non sur la fenêtre graphique
`forward(dist)` `backward(dist)`	avance ou recule de `dist` pixels dans la direction du crayon
`goto(x,y)`	déplacement du crayon au point de coordonnées `(x,y)`
`setx(x)` `sety(y)`	déplacement du crayon en ne modifiant que son abscisse ou son ordonnée
`right(angle)` `left(angle)`	la direction du tracé tourne de `angle` degrés vers la droite ou vers la gauche.
`setheading(angle)`	mise à jour de la direction du crayon (*orientation = cercle trigonométrique*)
`circle(R)`	trace un cercle de rayon R en partant de la position actuelle du crayon (≠ centre)

`reset()` réinitialisation de l'affichage : les tracés sont effacés, le crayon est placé au centre, …	`clear()` effacement du tracé sans bouger le crayon

`x,y=pos()`	permet de récupérer dans les variables `x,y` les coordonnées de la position du crayon
`xcor()` `ycor()`	retourne la valeur de l'abscisse ou de l'ordonnée du crayon
`heading()`	retourne la direction en degrés du crayon (*orientation = cercle trigonométrique*)

`width(n)`	mise à jour de l'épaisseur du tracé (`1` par défaut)
`color(coul)`	mise à jour de la couleur du crayon (`'black'` par défaut) coul peut être `'blue'`, `'red'`, `'yellow'`, `'brown'`, …

`screensize(L,H)`	mise à jour des dimensions de l'écran d'affichage. `L` et `H` sont les nouvelles demi-largeur et demi-hauteur. (`400` et `300` par défaut)
`dlarg,dhaut=screensize()`	permet de récupérer dans les variables `dlarg` et `dhaut` les demi-largeur et demi-hauteur de l'écran d'affichage
`speed(0)`	met la vitesse de tracé au maximum !