Résumé de Python

Quentin Fortier

September 21, 2021

Variables

= sert à modifier la valeur d'une variable

```
a = 7 # définition de a
b = a + 2 # définition de b
a = 3 # modification de a
```

On utilise == pour comparer deux valeurs :

```
a == b # renvoie False
```

Variables

Pour échanger 2 variables en Python :

```
a = 3
b = 7
a, b = b, a # échange a et b
```

Variables

Théorème

Si a, b sont deux entiers, il existe un unique q (**quotient**) et r (**reste**) tels que :

- $0 \leq r < b$
- $a = bq + r$

En Python :

- q est obtenu par `a // b`
- r est obtenu par `a // b`

```
7 // 3  # renvoie 2
7 % 3  # renvoie 1
```

Variables

a divise b

$\Longleftrightarrow$

le reste de la division de b par a est 0

$\Longleftrightarrow$

$b \,\%\, a == 0$

Variables

a divise b

$\Longleftrightarrow$

le reste de la division de b par a est 0

$\Longleftrightarrow$

$b \,\%\, a == 0$

```
def divise(a, b):
    return b % a == 0
```

Fonction

Définition de $f : x \longmapsto \sqrt{x} + 4x^2$:

```
def f(x):
    return x**0.5 + 4*x**2

f(1)  # donne 5
```

L'indentation (décalage obtenu avec la touche TAB), permet à Python de savoir ce qui est à l'intérieur de la fonction.

Condition if

```
if condition:      # condition doit être un booléen
    instructions   # (True ou False)
```

Condition if

```
def abs(x): # fonction valeur absolue
    if x < 0:
        return -x
    return x

abs(-5)  # renvoie 5
```

Condition if

On pourrait mettre un `else`, mais ce n'est pas obligatoire :

```
def abs(x): # fonction valeur absolue
    if x < 0:
        return -x
    else:
        return x
```

Boucles

Boucle for :

```
for i in range(6):
    print(i)
    # exécuté pour i = 0, i = 1, ..., i = 5
```

Boucles

Boucle for :

```
for i in range(6):
    print(i)
    # exécuté pour i = 0, i = 1, ..., i = 5
```

Exercice

Calculer $\sum_{k=1}^{10} k2^k$.

Boucles

Boucle for en spécifiant début et fin :

```
for i in range(3, 8):
    print(i)
    # exécuté pour i = 3, i = 4, ..., i = 7
```

Boucles

Boucle for en spécifiant début, fin et pas :

```
for i in range(6, 13, 2):
    print(i)
    # affiche les entiers pairs de 6 à 12
```

Boucles

Boucle while :

```
while condition:
    instructions
```

Attention : un while peut faire boucle infinie si la condition est toujours vraie

Boucles

Algorithme d'Euclide pour trouver le PGCD, en faisant des divisions euclidiennes :

```
def pgcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

Tuples

Un tuple (ou n-uplet) est similaire aux n-uplets en mathématiques (par exemple, un couple si $n = 2$) :

```
p = (1.3, 6.4) # 2-uplet de coordonnées 1.3 et 6.4
p[0]  # donne 1.3
len(p)  # donne 2
```

Un tuple peut servir à renvoyer plusieurs résultats par une fonction.

Tuples

Exercice

Écrire une fonction `milieu` telle que `milieu(p1, p2)` renvoie le milieu de `p1` et `p2` (des points dans le plan).

Listes

Une liste permet de stocker plusieurs éléments. Contrairement aux tuples, on peut ajouter un élément à la fin d'une liste avec `L.append(...)`

```
L = [-2, 4, 2.14]
L[1]  # donne 4 (élément d'indice 1)
L[-1] # donne 2.14 (dernier)
len(L) # donne 3
L.append(3)
# L vaut maintenant [-2, 4, 2.14]
```

Listes

Pour parcourir une liste, on parcourt ses indices :

```
for i in range(len(L)):
    print(L[i]) # affiche les éléments de L
```

Listes

Pour parcourir une liste, on parcourt ses indices :

```
for i in range(len(L)):
    print(L[i]) # affiche les éléments de L
```

Exercice

Écrire une fonction `appartient` telle que `appartient(e, L)` soit `True` si `e` appartient à `L`, `False` sinon.

Listes

Pour parcourir une liste, on parcourt ses indices :

```
for i in range(len(L)):
    print(L[i]) # affiche les éléments de L
```

Exercice

Écrire une fonction `appartient` telle que `appartient(e, L)` soit `True` si `e` appartient à `L`, `False` sinon.

Exercice

Écrire une fonction `inverse` renvoyant l'inverse d'une liste.

Listes

`L[i:j]` extrait une sous-liste d'une liste `L` des indices `i` (inclus) à `j` (exclu) :

```
L = [7, 0, 42, 21, 3, -5]
L1 = L[2:4]
#  L1 est la liste [42, 21]
```

`L[i:j]` effectue une copie de `L` : si on modifie `L1` cela ne modifie pas `L`.