Python
Les fonctions

Nicolas Delestre

INSA
ROUEN NORMANDIE

Définition standard d'une fonction 1 / 5

def

- Syntaxe :

```
def nom(param1, param2, ...):
    """ documentation """
    ...
```

- Il n'y a pas de procédure en python, uniquement des fonctions qui retournent `None` lorsque l'instruction `return` n'est pas utilisée ou utilisée avec aucun objet
- Le passage de paramètre est un passage par référence : possibilité de changer l'état de l'objet (si muable) mais il n'est pas possible de changer d'objet (référencé par le paramètre effectif)
- Une variable initialisée dans une fonction est une variable locale (sauf, si grâce à l'instruction `global`, elle est déclarée comme globale)
- Une variable utilisée sans être initialisée est considérée comme globale
- Lors de l'appel d'une fonction, l'association paramètres effectifs $\leftrightarrow$ paramètres formels peut être « positionnel » ou « nommé »

Exemple

```
def est_divisible_par(a, b):
    """ permet de savoir si a est divisible par b"""
    return a % b == 0
```

```
>>> est_divisible_par(39, 3)
True
>>> est_divisible_par(a=39, b=3)
True
>>> est_divisible_par(b=3, a=39)
True
```

```
>>> help(est_divisible_par)


Help on function est_divisible_par in module exemple:

est_divisible_par(a, b)
    permet de savoir si a est divisible par b
```

PEP 257

https://www.python.org/dev/peps/pep-0257/

INSA ROUEN NORMANDIE

- Une fonction peut définir des fonctions locales

Exemple

```
def est_premier(n):
    """permet de savoir si un nombre est premier ou pas"""

    def est_pair(n):
        return est_divisible_par(n, 2)

    if est_pair(n):
        return False
    else:
        for i in range(3, n // 2, 2):
            if est_divisible_par(n, i):
                return False
    return True
```

ROUEN NORMANDIE

- L'instruction `nonlocal` permet, dans une fonction locale, de déclarer qu'une variable est partagée par cette fonction et la fonction « mère »

Exemple

```
def toto():
    def tata():
        nonlocal x
        x = 1
    x = 0
    tata()
    return x
```

INSA ROUEN NORMANDIE

Annotation 1 / 2

def

- Syntaxe :

```
def nom(param1: annot1, param2: annot2, ...) -> annotRetour:
    """ documentation """
    ...
```

- Les annotations (à partir de python 3.4) sont optionnelles et doivent être des expressions python (souvent les types attendus, mais cela peut poser problème pour les classes en cours de définition)
- Les informations fournies par les annotations peuvent servir :
  - aux développeurs utilisateurs de la fonction (complète la documentation)
  - aux environnements de développement (où IDE)
  - à des vérificateurs statiques de type, tel que *mypy* (cf. `http://mypy-lang.org/`)

INSA ROUEN NORMANDIE

Exemple

```
def est_premier(n: int) -> bool:
    """permet de savoir si un nombre est premier ou pas"""
    if n % 2 == 0:
        return False
    else :
        for i in range(3, n // 2, 2):
            if n % i == 0:
                return False
    return True
```

En utilisant ≪ des valeurs ≫ par défaut

- Syntaxe :

```
def nom(param1, ..., paramOpt1 = val1, paramOpt2 = val2...):
    """ documentation """
    ...
```

- Les paramètres formels ayant une valeur par défaut deviennent optionnels
- Dès qu'un paramètre formel à une valeur par défaut, les suivants doivent aussi en avoir

Attention

Il est conseillé d'utiliser des objets immuable pour éviter tout problème

INSA ROUEN NORMANDIE

Définition de fonction d'arité variable 2 / 6

Exemple

```
def somme_naturels(fin, debut=1, pas=1):
    res = 0
    for i in range(debut,fin,pas):
        res = res + i
    return res
```

```
def somme_naturels(fin: int, debut: int=1, pas: int=1) -> int:
    res = 0
    for i in range(debut,fin,pas):
        res = res + i
    return res
```

```
>>> somme_naturels(10)
45
>>> somme_naturels(10,5)
35
>>> somme_naturels(10,pas=2)
25
>>> somme_naturels(debut=1,fin=10,pas=2)
```

En utilisant *args

- Syntaxe :

```
def nom(param1, ..., *args):
    """ documentation """
    ...
```

- Cela signifie que le nombre de paramètres effectifs non nommés peut être en nombre variable
- `*args` doit être déclaré après les paramètres formels nommés
- Lors de l'appel, il doit y avoir une valeur pour tous les paramètres nommés
- `args` est un tuple
- On peut transformer une séquence en une suite variable de paramètres effectifs grâce à l'opérateur `*`, par exemple avec `l = (1, 2, 3)`

  `foo(*l)` est équivalent à `foo(1, 2, 3)`

Définition de fonction d'arité variable 4 / 6

Exemple

```
def somme_nombres(debut, fin, *args):
    res = 0
    for i in args[debut:fin]:
        res = res + i
    return res
```

```
>>> somme_nombres(0,3,10,20,30,40,50)
60
>>> somme_nombres(0,3,10,20,30,40,50,60,70)
60
>>> somme_nombres(0,7,10,20,30,40,50,60,70)
280
>>> somme_nombres(0,3,*[10,20,30,40,50,60,70])
60
```

En utilisant `**kwargs`

- Syntaxe :

```
def nom(param1, ..., **kwargs):
    """ documentation """
    ...
```

- Cela signifie que le nombre de paramètres effectifs nommés peut être en nombre variable (différents des paramètres formels)
- `kwargs` est un dictionnaire
- `**kwargs` doit être déclaré en dernier, après `*args`
- On peut transformer un dictionnaire ayant des clés de type `str` en une suite de paramètres effectifs nommés grâce à l'opérateur `**`

ROUEN NORMANDIE

Définition de fonction d'arité variable 6 / 6

Exemple

```
1  def valeur_d_une_cle(dic, cle, valeur_si_non_present):
2      if dic:
3          if cle in dic:
4              return dic[cle]
5          else:
6              return valeur_si_non_present
7      else:
8          return valeur_si_non_present
9
10 def somme_nombres_v2(*args, **kwargs):
11     debut = valeur_d_une_cle(kwargs,'debut',0)
12     fin = valeur_d_une_cle(kwargs,'fin',len(args))
13     res = 0
14     for i in args[debut:fin]:
15         res = res + i
16     return res
```

```
>>> somme_nombres_v2(10,20,30,40)
100
>>> somme_nombres_v2(10,20,30,40,debut=1)
90
>>> somme_nombres_v2(10,20,30,40,debut=1,fin=3)
50
>>> somme_nombres_v2(10,20,30,40,fin=3)
60
```

Instruction `pass`

- Lors de la définition d'une fonction, la signature doit **obligatoirement** être suivie de la documentation et/ou du code
- L'instruction `pass` est une instruction qui ne fait rien
- Elle est utilisée lorsque l'on veut définir une fonction sans documentation qui ne fait rien (on reporte à plus tard son développement)

Exemple

```
def fonction_qui_ne_fait_rien():
    pass
```

INSA ROUEN NORMANDIE

Conclusion

Dans ce cours nous avons vu

- comment développer une fonction :
  - avec ou sans annotation,
  - en utilisant des variables locales ou globales, voire des variables non locales pour les fonctions locales
- comment appeler une fonction avec des liaisons paramètres effectifs $\leftrightarrow$ paramètres formels, positionnelles ou nommées
- comment développer des fonctions d'arité variable
- les opérateurs
  - `*` qui permet de transformer toute séquence en paramètres effectifs positionnels
  - `**` qui permet de transformer tout dictionnaire en paramètres effectifs nommés
- l'instruction `pass` qui permet de créer des fonctions « vides »

ROUEN NORMANDIE