Structures de contrôle

Test

Instruction if

```
if condition:
    bloc de lignes
else:
    bloc de lignes
```

Boucle

Instruction for..in

```
for variable in sequence:
    bloc de lignes
else:
    bloc de lignes
```

Instruction while

- L'instruction while permet d'exécuter un bloc de lignes tant qu'une expression est vérifiée.
- Si l'expression n'est plus vraie l'instruction else est exécutée si celle-ci existe et la boucle s'arrête.

```
while condition:
    bloc de lignes
else:
    bloc de lignes
```

Contrôle des itérations

continue	interrompt l'execution de la boucle pour l'élément en cours et passe à l'élement suivant
break	interrompt l'execution de la boucle et n'exécute pas l'instruction else

Les list comprehensions

Les lists comprehensions donnent un moyen concis d'appliquer une fonction sur chaque élément d'une liste afin d'en produire une nouvelle.

```
>>> nombres = [1,3,8,9,7,32,89,42]
>>> # retourner sous forme de liste chacun des nombres au carré
>>> [nombre*nombre for nombre in nombres]
[1, 9, 64, 81, 49, 1024, 7921, 1764]

>>> nombres = [1,3,8,9,7,32,89,42]
>>> # retourner sous forme de liste les nombres pairs
>>> [nombre for nombre in nombres if nombre%2==0]
[8, 32, 42]
```

Les dict comprehensions

Les dict comprehensions donnent un moyen concis de créer un dictionnaire à partir d'une itération.

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

Exceptions

try..except..else

Lorsqu'une exception est levée il est possible de la stopper en interceptant l'erreur.

```
try:
    bloc de code surveillé
except:
    bloc de code exécuté si l'exception est levée
```

Il est fortement conseillé de spécifier l'exception afin de ne pas masquer silencieusement tous les types d'erreur qui pourrait intervenir.

```
try:
    bloc de code surveillé
except Exception:
    bloc de code exécuté si l'exception est levée
```

Exemples d'exceptions

- AttributeError
- ImportError
- IndentationError
- IndexError
- KeyError
- MemoryError
- SyntaxError
- TypeError
- ZeroDivisionError

try..finally

```
try:
    bloc de code surveillé
finally:
    code toujours exécuté (y compris si raise ou return dans le bloc surveillé).
```

Fonctions

- La définition d'une fonction se fait par le mot clé def suivi du nom de la fonction.
- Les éventuels paramètres sont déclarés entre parenthèses.
- Le caratère ":" délimite le début de la fonction.
- Le bloc de la fonction est délimité par les indentations.

Définition

```
def say_hello(firstname, lastname):
    print ("Hello {} {}".format(firstname.title(), lastname.title()))
```

Utilisation

```
say_hello('Terry', 'gilliam')
```

Paramètres

Paramètres explicites

- Les paramètres explicites sont séparés par une virgule et peuvent être enrichis d'une valeur par défaut, ils sont dans ce cas optionnels.

- Il est nécessaire de regrouper les paramètres optionnels à la fin de la liste des paramètres.
- Les paramètres peuvent être nommés sans respecter un ordre précis.

```
>>> def say_hello(firstname, gender='Mr', lastname=''):
...     fullname = ' '.join([gender, firstname, lastname])
...     print("Hello {}".format(fullname))

>>> say_hello('Terry')
Hello Mr Terry
>>> say_hello('Yolande', 'Mme')
Hello Mme Yolande
>>> say_hello('Yolande', lastname='Moreau', gender='Mme')
Hello Mme Yolande Moreau
```

Paramètres non explicites

- Les paramètres non explicites permettent de spécifier autant de valeurs que l'on souhaite sans qu'il soit nécessaire de les spécifier dans la signature de la fonction.
- Ils sont fournis sous la forme nom=valeur, et sont accessibles à l'intérieur de la fonction sous forme de dictionnaire.
- Les paramètres non explicites sont préfixés d'une double étoile.

```
>>> def show_actors(**actors):
...     for actor, name in actors.items():
...         print("{}:{}" .format(actor, name))

>>> show_actors(actor1='Terry Gilliam', actor2='John Cleese')
actor2:John Cleese
actor1:Terry Gilliam
```

Paramètres arbitraires

- Les paramètres arbitraires fonctionnent de la même manière que les paramètres non explicites mais ne sont pas nommés.
- Il sont accessibles à l'intérieur de la fonction sous forme d'un tuple.
- Les paramètres arbitraires sont préfixés d'une étoile

```
>>> def ajoute_acteurs(*actors):
...     liste = []
...     for actor in actors():
...         liste.append(actor)
...     return liste

>>> ajoute_cteurs('Terry Gilliam', 'John Cleese')
['John Cleese', 'Terry Gilliam']
```

Les 3 types de paramètres peuvent être utilisés dans une même fonction, dans ce cas leur ordre doit être le suivant:

```
def fonction(param1, param2=2, *arbitraires, **non-explicites):
```

Docstrings

Les docstrings sont des chaines de caractères placées au début du corps des fonctions. Ils sont associés à la variable doc de la fonction.

```
>>> def get_full_name(firstname, lastname):
...     """Retourne le nom complet"""
...     return ' '.join([firstname, lastname])

>>> get_full_name.__doc__
'Retourne le nom complet'
```
