INF8007 – Languages de scripts
Syntaxe de base

Antoine Lefebvre-Brossard

Hiver 2018

Exemple de programme


```
#!/usr/bin/python
def fibonacci(n):
    x1, x2 = 1, 1
    if n == 1 or n == 2:
        return 1
    else:
        for _ in range(2, n):
            x1, x2 = x2, x1 + x2
        return x2
print(fibonacci(3))
>>> 2
print(fibonacci(5))
>>> 5
print(fibonacci(10))
>>> 55
```

- Utilise des espaces pour définir la syntaxe
- Le type des variables n'a pas à être défini
- L'assignation des variables peut être de plusieurs façons
- Le début d'un bloc est défini par ":" et celui-ci est défini par son indentatioin

Variables


```
x = "abc"                # str
y = 'abc'                # str
x = 10                   # int
y = 5.3                  # float
x = .3                   # float
y = 5.                   # float
x = [1, 2, 3]            # list
y = (1, 2, 3)            # tuple
x = {1, 2, 3}            # set
y = {a: 1, b: 2, c: 3} # dict
x, y = "a", 1            # str et int
```

- Possible de changer le type d'une variable en lui assignant une nouvelle valeur
- Différente façons de définir une même valeur

list

- Contient un nombre variable d'éléments de type variable

```
a = [1, "a", [1, 2, 3]]
```

- Plusieurs façons d'ajouter des éléments
  - `a.append("a")`
  - `a = a + ["a"]`
  - `a = ["a"] + a`
  - `a.insert(0, "a")`
- Peut être additionné ou multiplié
  - `[1, 2, 3] + [4, 5, 6]`

    `>> [1, 2, 3, 4, 5, 6]`
  - `[1, 2] * 3`

    `>> [1, 2, 1, 2, 1, 2]`

dict

- Paires de clé-valeur
- La clé doit être unique, mais la valeur peut être n'importe quoi

```
a = {1: [1, 2, 3], 2: {"a": 10, "b": 100}}
```

- Deux façons d'ajouter un élément
  1. `a[3] = 5`
  2. `a.update({3: 5})`
- Utile pour représenter des données de façons similaire à JSON

```
data = [{"name": "alice", "age": 23},
        {"name": "bob"  , "age": 25}]
```

set

- Représente un ensemble de valeurs uniques
- Non ordonné
- Utilise les opérateurs "`|`" (union), "`&`" (intersection) et "`-`" (différence)
- Utile pour trouver les éléments uniques d'une liste

```
set([1, 2, 3, 2, 4])
>> {1, 2, 3, 4}
```

for

- Possède la structure

```
for `variable` in `collection`:
    # do something...
```

- La collection la plus utilisé est `range([min],max,[step])` qui correspond au `for` classique d'autres languages
- Très utile avec la fonction `enumerate` pour avoir à la fois l'élément et son index

```
for i, x in enumerate(["a", "b", "c"]):
    print(f"The {i}th element is {x}.")
>> The 0th element is a.
>> The 1th element is b.
>> The 2th element is c.
```

while

- Attend une condition avant de finir
- Possède la structure

```
while `condition`:
    # do somthing...
```

- Meilleur moyen d'avoir un bug dans son programme si la condition n'est jamais remplie
- `break` peut aussi être utilisé pour arrêter la boucle (fonctionne aussi avec les boucles for)

if

- Possède la strucure :

```
if `condition`:
    # do something...
elif `other condition`:
    # do something else...
else:
    # do this other thing...
```

- Peut aussi être utilisé en une ligne, mais est en général peu recommandé sauf dans des cas particuliers

```
a = [b] if isinstance(b, int) else b
```

Fonctions

- Structure de base

```
def fct(a, b, c=5):
    """This is a docstring for
    the function `fct`
    """
    # do something that gives a variable `d`...
    return d
```

- Débute par `def`
- Les paramètres avec défaut doivent après ceux sans
- Peut posséder des types (falcutatifs) pour mieux comprendre le code. Ceux-ci ne changent pas la performance

```
def fct(a: int, b: bool, c: int=5) -> str:
    # do something that gives a variable `d`...
    return d
```

Objets

- Structure de base

```
class Obj(object):
    def __init__(self, a):
        self.a = a
    def fct(self):
        return self.a * 2
o = Obj(5)
```

- L'objet `Obj` hérite de `object` par défaut et `(object)` est optionnel
- Tout ce qui se passe à l'initialisation est dans `__init__`
- Toute méthode de l'object doit posséder comme première variable `self` (ce nom est celui qui est toujours utilisé)
- Sujet beaucoup plus large, mais l'important est que chaque instance d'un oject possède des attributs (`a` en haut) et des méthodes (`fct` en haut)

Capture des erreurs

- Structure de base

```
try:
    # main stuff to try to execute...
except `SomeError` e:
    # manage error...
finally:
    # always execute this...
```

- Est utilisé lorsqu'il y a une erreur et que l'on veut que le programme gère lui-même cette erreur (en la loggant ou en exécutant un code différent si elle arrive)