Méthodologie de programmation

Session 3

Alex Singh

Les fonctions en tant qu'abstractions

- Lors de la programmation, nous nous retrouvons souvent à réutiliser des morceaux de code.

Les fonctions en tant qu'abstractions

- Lors de la programmation, nous nous retrouvons souvent à réutiliser des morceaux de code.
- Il est pratique de regrouper (= abstraire) ces « sous-programmes » dans des objets réutilisables appelés fonctions.

Les fonctions en tant qu'abstractions

- Lors de la programmation, nous nous retrouvons souvent à réutiliser des morceaux de code.
- Il est pratique de regrouper (= abstraire) ces « sous-programmes » dans des objets réutilisables appelés fonctions.
- Étant donné que ces morceaux de code réutilisables peuvent dépendre d'entrées externes, les fonctions peuvent avoir des *arguments*.

Les fonctions en tant qu'abstractions

- Lors de la programmation, nous nous retrouvons souvent à réutiliser des morceaux de code.
- Il est pratique de regrouper (= abstraire) ces « sous-programmes » dans des objets réutilisables appelés fonctions.
- Étant donné que ces morceaux de code réutilisables peuvent dépendre d'entrées externes, les fonctions peuvent avoir des *arguments*.
- Étant donné que ces morceaux de code réutilisables peuvent produire des résultats qui seront utilisés ultérieurement, les fonctions peuvent *renvoyer* des données (si elles ne sont pas spécifiées par l'utilisateur, alors `None`).

Fonctions en Python

- Définition et appel d'une fonction en Python :

```
def f(x):
  y = 2
  return x+y
f(5)
f(1+2+3)
f("hi") #?
```

Fonctions en Python

- Définition et appel d'une fonction en Python :

```
def f(x):
  y = 2
  return x+y
f(5)
f(1+2+3)
f("hi") #?
```

- Les arguments multiples sont séparés par des virgules :

```
def f(x,y,z):
  return x+2*y+3*z
f(1,2,3), f(1,2)
```

Fonctions en Python

- Les arguments peuvent avoir des valeurs par défaut :

```
def f(x,y,z=0):
  return x+2*y+3*z
f(1,2,3)
f(1,2)
```

- Les arguments passés lors des appels peuvent être *positionnels* ou *nommés*, tous les arguments positionnels apparaissant avant les arguments nommés.

```
f(1,2,3)
f(1,y=2,z=3)
f(x=1,z=3,y=2)
f(x=1,2,3)
```

Fonctions

- Les fonctions peuvent modifier le « contenu » de leurs arguments (s'ils sont modifiables).

```
def f(x):
  x = x + 1
x = 1
f(x)
```

par opposition à

```
def f(l):
  l.append(1)
l = []
f(l)
```

Fonctions en Python

- Optez pour des noms de variables, d'arguments et de fonctions descriptifs et lisibles.
- Préférez passer des arguments nommés, en particulier lorsqu'ils sont nombreux : « mieux vaut être explicite qu'implicite » (Zen of Python).
- Organisez votre code autour de fonctions (et plus tard, peut-être, d'objets).
- Soyez conscient de la portée et de la mutabilité.