Méthodologie de programmation

Session 2

Alex Singh

Ingrédients pour la programmation

- La plupart des langages de programmation fournissent trois éléments fondamentaux :
  - Expressions primitives.

Ingrédients pour la programmation

- La plupart des langages de programmation fournissent trois éléments fondamentaux :
  - Expressions primitives.
  - Moyens de combinaison.

Ingrédients pour la programmation

- La plupart des langages de programmation fournissent trois éléments fondamentaux :
  - Expressions primitives.
  - Moyens de combinaison.
  - Moyens d'abstraction.

Ingrédients pour la programmation

- La plupart des langages de programmation fournissent trois éléments fondamentaux :
  - Expressions primitives.
  - Moyens de combinaison.
  - Moyens d'abstraction.
- En Python, ceux-ci se présentent comme suit :
  - Expressions primitives : `0`, `1.2+1`, `True`, `True || False`, `[1,2,3]`, `"hi!"`, `[“hi”,"bonjour",[3]]` ...

Ingrédients pour la programmation

- La plupart des langages de programmation fournissent trois éléments fondamentaux :
  - Expressions primitives.
  - Moyens de combinaison.
  - Moyens d'abstraction.
- En Python, ceux-ci se présentent comme suit :
  - Expressions primitives : `0`, `1.2+1`, `True`, `True || False`, `[1,2,3]`, `"hi!"`, `[“hi”,"bonjour",[3]]` ...
  - Moyens de combinaison : `if`, `for`, `while`...
  - Moyens d'abstraction : `=`, `def`, `class`...

Du code source à l'exécution

- Un programme est un ensemble d'instructions à exécuter par l'ordinateur.

Du code source à l'exécution

- Un programme est un ensemble d'instructions à exécuter par l'ordinateur.
- Sous sa forme lisible par l'homme, il est souvent appelé « code source ».

Du code source à l'exécution

- Un programme est un ensemble d'instructions à exécuter par l'ordinateur.
- Sous sa forme lisible par l'homme, il est souvent appelé « code source ».
- L'exécution se fait souvent par le biais (d'une combinaison) d'interprétation et de compilation.

Du code source à l'exécution

- Un programme est un ensemble d'instructions à exécuter par l'ordinateur.
- Sous sa forme lisible par l'homme, il est souvent appelé « code source ».
- L'exécution se fait souvent par le biais (d'une combinaison) d'interprétation et de compilation.
- Compilation : traduction d'un langage source vers un langage cible (généralement de niveau inférieur).

Du code source à l'exécution

- Un programme est un ensemble d'instructions à exécuter par l'ordinateur.
- Sous sa forme lisible par l'homme, il est souvent appelé « code source ».
- L'exécution se fait souvent par le biais (d'une combinaison) d'interprétation et de compilation.
- Compilation : traduction d'un langage source vers un langage cible (généralement de niveau inférieur).
- Interprétation : un interpréteur exécute directement le code.
- CPython (l'implémentation de référence) utilise une combinaison : compilation en bytecode, interprétation via une machine virtuelle.

Python

- Python est un langage de *haut niveau* : il fait abstraction des détails spécifiques à la machine.

Python

- Python est un langage de *haut niveau* : il fait abstraction des détails spécifiques à la machine.
- Python est un langage *polyvalent* : il est conçu pour créer des logiciels largement applicables.

Python

- Python est un langage de *haut niveau* : il fait abstraction des détails spécifiques à la machine.
- Python est un langage *polyvalent* : il est conçu pour créer des logiciels largement applicables.
- Python comprend plusieurs *paradigmes* : impératif, orienté objet, et un peu fonctionnel.

Python

- Python est un langage de *haut niveau* : il fait abstraction des détails spécifiques à la machine.
- Python est un langage *polyvalent* : il est conçu pour créer des logiciels largement applicables.
- Python comprend plusieurs *paradigmes* : impératif, orienté objet, et un peu fonctionnel.
- Les données sont représentées par des *objets* qui ont un *type* et une *valeur*.

Python

- Python est un langage de *haut niveau* : il fait abstraction des détails spécifiques à la machine.
- Python est un langage *polyvalent* : il est conçu pour créer des logiciels largement applicables.
- Python comprend plusieurs *paradigmes* : impératif, orienté objet, et un peu fonctionnel.
- Les données sont représentées par des *objets* qui ont un *type* et une *valeur*.
- Les contraintes de type sont vérifiées de manière dynamique (pendant l'exécution).

Python

- Python est un langage de *haut niveau* : il fait abstraction des détails spécifiques à la machine.
- Python est un langage *polyvalent* : il est conçu pour créer des logiciels largement applicables.
- Python comprend plusieurs *paradigmes* : impératif, orienté objet, et un peu fonctionnel.
- Les données sont représentées par des *objets* qui ont un *type* et une *valeur*.
- Les contraintes de type sont vérifiées de manière dynamique (pendant l'exécution).
- Syntaxe lisible, espaces blancs comme délimiteurs pour les blocs.

Expressions primitives : constantes

- Données numériques :
  - entiers : `-1, 0, 1, 42, +, *, /, //`
  - flottants : `42.0`
  - nombres complexes : `1+1j`
- Booléens : `True, False`
- Séquences :
  - Liste : `[1,2,3], [[],[4,[5],"a"],2j]`
  - Tuple : `(1,2), (1,2,3,4)`
  - Plage : `range(0,10)`
- Chaînes (type séquence de texte) : `"c", 'c', "Hello world!""`
- Autres : `bytes, bytearray`...

Expressions primitives : opérations de base

- Pour les données numériques : `+`,`-`,`*`,`/`,`//`,`%`,`**`,`abs()` et fonctions de conversion (`int()`, `float()`, `complex()`).
- Pour les booléens : `and`, `or`, `not`.
- Pour les séquences :
  - Accès via index/tranches : `l[i:j:k]`
  - Tests d'appartenance : `2 in [1,2]` ,
  - Concaténation : `[1] + [2,3]` ,
  - Longueur : `len(l)`
- Fonctionnalités supplémentaires fournies par la bibliothèque standard Python.

Expressions composées

- Exécution conditionnelle de blocs de code :

```
if expr1:
    suite
elif expr2:
    suite
else:
    suite
```

- Les expressions `expr1, expr2` doivent être interprétables comme des valeurs booléennes :
  - `False, None`, le zéro numérique et les séquences vides sont tous « faux ».
  - Toutes les autres valeurs sont « vraies ».
- Zero ou plusieurs instructions `elif` peuvent être présentes.

Expressions composées: iteration

- Répéter un bloc tant qu'une condition est vraie :

```
while expr:
    suite
```

- Ça boucle tant que `expr` reste « vrai ».

Expressions composées: iteration

- Répéter un bloc tant qu'une condition est vraie :

```
while expr:
    suite
```

- Ça boucle tant que `expr` reste « vrai ».
- Itération à travers les éléments d'un itérable :

```
for x in l:
    suite
```

- L'expression `x` peut être un identifiant ou même une séquence.
- L'expression `l` doit correspondre à un itérable (tel qu'une valeur de type séquence).

Expression composée : fonctions

- Définition d'une fonction :

```
def f(args):
    suite
    return expr
```

- Les instructions de retour sont facultatives et une fonction renvoie implicitement `None` par défaut.
- Application d'une fonction : `f(x)` .
- Les fonctions sont un exemple prototypique d'abstraction.

Variables en Python

- Nous affectons des données à des variables. Par exemple

```
x = 3
print(x) #affiche 3
```

Variables en Python

- Nous affectons des données à des variables. Par exemple

```
x = 3
print(x) #affiche 3
```

- Les variables en Python sont mutables : les données qu'elles contiennent peuvent changer.

```
x = 3
print(x)
x = "Salut !"
print(x)
```

Variables en Python

- Nous affectons des données à des variables. Par exemple

```
x = 3
print(x) #affiche 3
```

- Les variables en Python sont mutables : les données qu'elles contiennent peuvent changer.

```
x = 3
print(x)
x = "Salut !"
print(x)
```

- L'*état* d'un programme à un moment donné est l'ensemble des données et des variables. Il peut évoluer de manière complexe et difficile à prévoir.

Variables en Python

- Nous affectons des données à des variables. Par exemple

```
x = 3
print(x) #affiche 3
```

- Les variables en Python sont mutables : les données qu'elles contiennent peuvent changer.

```
x = 3
print(x)
x = "Salut !"
print(x)
```

- L'*état* d'un programme à un moment donné est l'ensemble des données et des variables. Il peut évoluer de manière complexe et difficile à prévoir.

Portée

- Les variables ont une portée : une partie du programme dans laquelle elles sont valides et accessibles.

Portée

- Les variables ont une portée : une partie du programme dans laquelle elles sont valides et accessibles.

```
x = 1
for i in range(0,10):
  y = 2
  print(x, y, i)
print(x,y,i)
```

- La portée peut être : globale ou locale (limitée à une fonction, une classe, un module, une compréhension de liste, etc.).

Portée

- Les variables ont une portée : une partie du programme dans laquelle elles sont valides et accessibles.

```
x = 1
for i in range(0,10):
  y = 2
  print(x, y, i)
print(x,y,i)
```

- La portée peut être : globale ou locale (limitée à une fonction, une classe, un module, une compréhension de liste, etc.).
- À retenir : déterminez où une variable sera utilisée et définissez-la dans la portée appropriée.