Python
Les annotations (ou *Type hints*)

Nicolas Delestre

INSA ROUEN NORMANDIE

Rappels 1 / 3

Typage statique, typage dynamique

- Typage statique : un paramètre formel ou une variable (pour une portée donnée) est associé à un type. Le compilateur peut vérifier si les paramètres effectifs ou les affectations, respectent cette déclaration
- Typage dynamique : le type d'un paramètre formel ou d'une variable, n'est pas associé à un type, son type peut varier au cours du temps

Le Python adopte de typage dynamique

INSA ROUEN NORMANDIE

Rappels 2 / 3

Documentation formelle et informelle

- Documentation : information sur le code non prise en compte par le compilateur ou l'interpréteur
- Dans la plupart des langages, la documentation est présente dans des commentaires en suivant une certaine syntaxe (javadoc, doxygen, robotdoc, etc.)

Ne pas confondre documentation et commentaire

Cas de Python

- Documentation informelle : les docstring (cf. PEP 257)
- Documentation formelle : les annotations (cf. PEP 484)

INSA ROUEN NORMANDIE

Rappels 3 / 3

Annotation

- Disponible depuis la version 3.5
- Documentation formelle associée à une entité (paramètre formelle, variable, valeur retournée par une fonction, etc.) qui spécifie le type de cette entité
- Elle peut être utilisée par :
    - l'IDE pour aider le développeur lors du développement
    - des programmes de vérification comme pyright et mypy
- Elle n'est pas utilisée par l'interpréteur Python

Exemple

```python
1 def hello_world(ch: str) -> str:
2     return "Bonjour " + ch
```

Collections de base

- Il est possible de spécifier le type des objets que doit contenir une collection (**tuple**, **list** , **set**, **dict**)
  - avant la version 3.9, utilisation des déclarations `Tuple`, `List`, `Set`, `Dict` du module `typing` (*deprecated*)
  - après la version 3.9 directement les types **tuple**, **list** , **set**, **dict**

Exemple

```python
def somme_des_nombres_pairs(nombres: list[int]) -> int:
    return sum([nombre for nombre in nombres if nombre % 2 == 0])
```

INSA ROUEN NORMANDIE

Union et Optional

- Union (ou, à partir de la version 3.10, l'opérateur barre |) permet de déclarer plusieurs types possibles pour une annotation

- Optional permet d'indiquer que le type de l'entité peut être aussi None

Exemple

```
1 from typing import Union, Optional
2
3 def somme(data: Union[list[int], np.ndarray]) -> Optional[int]:
4     If isinstance(data, list):
5         return sum(data)
6     if isinstance(data, np.ndarray):
7         return int(np.sum(data))
8     return None
9
10 def somme(data: list[int]|np.ndarray]) -> Optional[int]:
11     ...
```

Self

- Lorsque l'on déclare une méthode de classe qui prend en paramètre ou qui retourne un objet du type de la classe que l'on est en train de définir, on ne peut pas utiliser son identifiant :
  - Avant Python 3.10, on utilisait la chaîne de caractères identique à l'identifiant de la classe
  - À partir de Python 3.10, on utilise `Self` du module `typing`

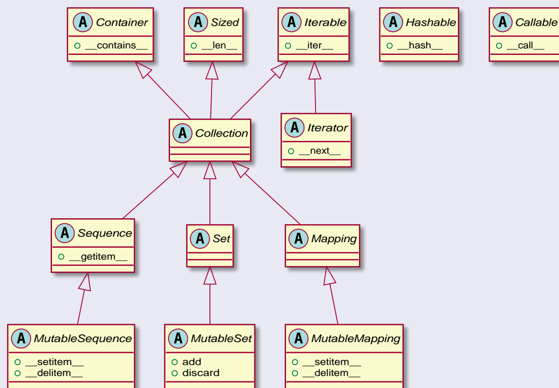
Exemple (extrait de la documentation de Python)

```
1  class Foo:
2      def return_self(self) -> "Foo":
3          ...
4          return self
5
6  from typing import Self
7  class Foo:
8      def return_self(self) -> Self:
9          ...
10         return self
```

Des types génériques

- Plutôt que de spécifier que le paramètre formel d'une fonction doit être un **tuple**, une **list** , un **set**, etc. on peut aussi spécifier que l'on a juste besoin d'itérer sur ce paramètre et donc qu'il est du type `Iterable`

Des types génériques proposés par le module `typing` (issus de `collection.abc`)



Les alias

- Un alias de type permet d'alléger et donner du sens à des annotations complexes
- Créé à l'aide de l'instruction **type** (à partir de 3.12)
- Un alias est une instance de la classe `TypeAliasType`

Exemple (extrait de la documentation de Python)

Plutôt que d'avoir :

```
1 def broadcast_message(
2         message: str,
3         servers: Sequence[tuple[tuple[str, int], dict[str, str]]]) -> None:
4     ...
```

On utilise les alias :

```
1 type ConnectionOptions = dict[str, str]
2 type Address = tuple[str, int]
3 type Server = tuple[Address, ConnectionOptions]
4
5 def broadcast_message(message: str, servers: Sequence[Server]) -> None:
```

La généricité

- Lorsqu'une classe stocke des éléments de même type, il est possible de paramétrer cette classe

```
1  class File[T]:
2      def __init__(self, *args: T):
3          self._elements = list(args)
4
5      def defiler(self) -> T:
6          return self._elements.pop(0)
7
8      def enfiler(self, element: T) -> None:
9          self._elements.append(element)
10 ...
```

- Avant la version 3.11, il fallait déclarer `T` :

```
T = TypeVar('T')
```

Conclusion

- À partir de la version 3.5, Python propose d'annoter classes, paramètres formels, variable et valeur de retour de fonction
- Ces informations formelles sont utilisées par les IDE et des programmes de vérification de type statique (par exemple pyright et mypy)
- Ces informations ne sont pas utilisées par l'interpréteur Python
- L'utilisation des annotations permet d'améliorer la qualité du code

INSA ROUEN NORMANDIE