UE12 P24 - Python Variables, types, fonctions

Objectif

Comprendre comment bien représenter les données dans nos programmes.

Variables

- Les variables représentent les données de nos programmes
- Les variables sont associées à des types
  - En Python, une variable peut changer de type
  - Mais c'est rarement une bonne idée (c'est plus compliqué d'identifier le type à un instant donné)

```
1 my_var = 1
2 print(type(my_var))
```

```
<class 'int'>
```

Variables

Une mauvaise idée

```
1 my_var = 'abc'
2 print(type(my_var))
3 my_var = 123
4 print(type(my_var))
```

```
<class 'str'>
<class 'int'>
```

Une meilleure idée

```
1 my_str_var = 'abc'
2 print(type(my_str_var))
3 my_int_var = 123
4 print(type(my_int_var))
```

```
<class 'str'>
<class 'int'>
```

Important

- Le plus important dans votre code : la lisibilité
- La machine comprendra toujours votre code, les autres développeurs non
  - Pensez à vous quand vous reprendrez votre code dans 6 mois

Variables et portée

- Une variable a une portée
- Variables globales / locales

> **Tip**
>
> Utilisez le moins possible de variables globales

```python
my_var = 1

def test(my_var='abc'):
    print(my_var)
    my_var = True
    print(my_var)

def test2():
    my_var = []
    print(my_var)

def test3():
    print(my_var)

test('def')
test2()
test3()
```

```
def
True
[]
1
```

Variables, égalité, identité

- Une variable pointe vers une case mémoire
- Il y a identité si 2 variables pointent vers la même case : `my_var_1 is my_var_2`
- Il y a égalité si 2 variables ont la même valeur : `my_var_1 == my_var_2`

Types simples :

```
1 var_1, var_2 = 1, 1
2 print(var_1 is var_2)
3 print(var_1 == var_2)
4 print(id(var_1), id(var_2))
```

```
True
True
140724576130720 140724576130720
```

Tous les autres types :

```
1 var_1, var_2 = [1, 2, 3], [1, 2, 3
2 print(var_1 is var_2)
3 print(var_1 == var_2)
4 print(id(var_1), id(var_2))
```

```
False
True
2039964588992 2039964589504
```

Variables et références

> ⚠ **Warning**
>
> Si `a is b`, modifier a modifie b

```
1 my_list_1 = [1, 2, 3]
2 my_list_2 = my_list_1
3 my_list_2.append(4)
4 print(my_list_1, my_list_2)
```

```
[1, 2, 3, 4] [1, 2, 3, 4]
```

Variables et références

Ca peut paraître évident, mais ça ne l'est pas toujours. Par exemple si vous voulez représenter une grille ou une matrice :

```
>>> l = [[0, 0]] * 2
>>> l
[[0, 0], [0, 0]]
>>> l[0][0] = 1
>>> l
[[1, 0], [1, 0]]
```

Variables et références

Mais on pense parfois modifier un objet, alors qu'on en crée un nouveau !

Pas de nouvel objet :

```
1 my_list_1 = [1, 2, 3]
2 my_list_2 = my_list_1
3 my_list_2.append(4)
4 print(my_list_1, my_list_2)
```

```
[1, 2, 3, 4] [1, 2, 3, 4]
```

Création d'un nouvel objet :

```
1 my_list_1 = [1, 2, 3]
2 my_list_2 = my_list_1
3 my_list_2 += [4]
4 print(my_list_1, my_list_2)
```

```
[1, 2, 3, 4] [1, 2, 3, 4]
```

Types

Vous connaissez déjà les types de bases :

Numériques :

```python
empty_var: None = None
bool_var: bool = True
int_var: int = 1
float_var: float = 1.0
```

Caractères et octets :

```python
my_str: str = 'Salut 👋'
my_bytes: bytes = '\x01\x02'
```

Collections :

```python
list_var: 'list[int]' = [1, 2, 3]
dict_var: 'dict[int, str]' = {1: 'ab', 2: 'cd'}
set_var: 'set[str]' = {'mines', 'paris'}
```

Classes

Vous pouvez définir vos propres types d'objets :

```python
class User:
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

def greet_user(user: User) -> None:
   print(f'Salut {user.name} !')
```

Fonctions et méthodes

Vous pouvez définir des fonctions et des méthodes, pensez bien à les typer.

```python
class Message:
    def __init__(self, sender: str, recipient: str, content: str):
        self.sender = sender
        self.recipient = recipient
        self.content = content
        self.time: datetime = datetime.now()

    def preview(self) -> str:
        return f'{self.sender} to {self.recipient} at {self.time}: {self.co

def get_latest_message(message_list: 'list[Message]') -> Message:
    ...
```

Typage et mensonges

Vous pouvez mentir dans vos annotations :

```python
def get_list_len(my_list: 'list[bool]') -> int:
    return len(my_list)

print(get_list_len({'a': 1, 'b': None}))
```

```
2
```

> **Caution**
>
> Ca signifie que votre IDE ne vous préviendra pas si vous vous trompez

> **Note**
>
> Mais ça vous aidera quand même à débugguer

Résumé

- Typez le plus possible vos fonctions et variables
- Les annotations de types sont “récents” et optionnels en Python
  - Elles sont souvent absentes des exemples en ligne : pensez à les rajouter !
- C’est une documentation importante pour vous
- Une variable peut changer de type : ne le faites pas