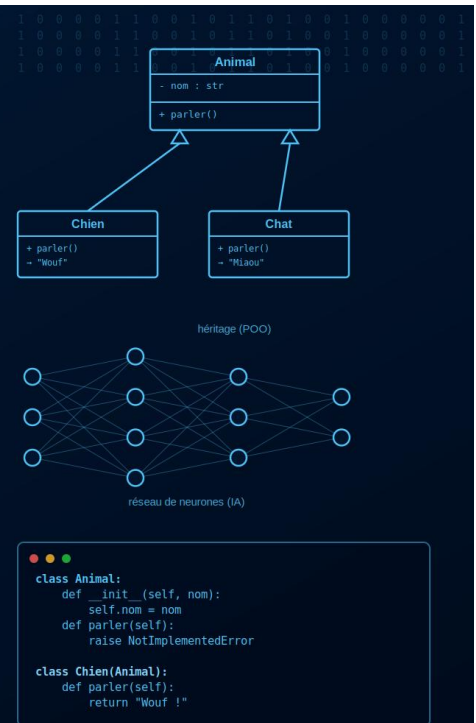


II. Bases du langage



M. Martin

1

1

Python propose plusieurs types fondamentaux :

- Nombres entiers : **int** (pas de limite de taille)
- Nombres réels : **float** (double précision)
- Nombres complexes : **complex**
- Booléens : **True, False** (héritent du type int)
- Chaînes de caractères : **str** (séquences immuables de caractères)
- Nones : valeur spéciale **None** (absence de valeur)

Exemples :

```
a, b, c, d = 42, 0b010, 0o542, 0xE5 # entiers (décimal, binaire, octal, hexa)
e, f = 3.14, -6.92e-6 # flottants
z = 2 + 3j # complexes
print(z * z.conjugate()) # 13 + 0j
c = True # booléen
s = "Python" # chaîne
n = None # objet spécial
```

2

2

- Arithmétiques : +, -, *, /, //, %, **
- Comparaisons : ==, !=, <, <=, >, >=
- Logiques : and, or, not
- Binaires : &, |, ^

Exemples :

```
3 + 2 # 5
7 // 2 # 3 ( division entière )
7 % 2 # 1 ( reste de la division )
2 ** 10 # 1024

x = 5
0 < x < 10 # True ( comparaisons enchaînées )
```

3

3

Les chaînes sont des séquences immuables :

```
s = "Python"
s[0] # 'P'
s[-1] # 'n'
s[1:4] # 'yth' (tranchage)
```

Concaténation et répétition

```
"Py" + "thon" # 'Python'
"ha" * 3 # 'hahaha'
```

Fonctions et méthodes utiles

```
len(s) # 6
s.upper() # 'PYTHON'
s.lower() # 'python'
s.replace("Py", "Ru") # 'Ruthon'
"abc,def".split(",") # ['abc', 'def']
```

4

4

Formatage et f-strings

```
x = 3.14159  
print(f"Valeur de pi = {x:.3f}") # 'Valeur de pi = 3.142'
```

5

5

Liste (list)

- Séquence mutable
- Supporte : ajout, suppression, tranchage, compréhension

```
l = [1, 2, 3]  
l.append(4) # [1, 2, 3, 4]  
l[1:3] # [2, 3]  
l[:2] # [1, 3]
```

Tuple (tuple)

- Séquence immuable
- Utilisé pour des collections fixes ou comme clés de dictionnaire

```
t = (1, 2, 3)
```

6

6

Ensemble (set)

- Collection non ordonnée et sans doublons

```
s = {1, 2, 2, 3}
print(s) # [1, 2, 3]
```

Dictionnaire (dict)

- Paire clé-valeur

```
d = {"nom": "Fulgence", "age": 25}
d["nom"] # 'Fulgence'
d["age"] = 26 # modification
for k, v in d.items(): # 'nom : Fulgence'
    print(k, " : ", v) # 'age : 26'
print(d.get("ville", "inconnu")) # 'inconnu'
```

7

7

Les compréhensions fournissent une syntaxe compacte pour générer des listes, des ensembles ou des dictionnaires.

Liste en compréhension

```
squares = [x**2 for x in range(6)] # [0, 1, 4, 9, 16, 25]
```

Avec condition :

```
pairs = [x for x in range(10) if x % 2 == 0] # [0, 2, 4, 6, 8]
```

Ensemble en compréhension

```
vowels = {c for c in "abracadabra" if c in "aeiou"} # {'a'}
```

Dictionnaire en compréhension

```
squares = {x: x ** 2 for x in range(4)} # {0: 0, 1: 1, 2: 4, 3: 9}
```

8

8

Conditions

```
x = 5
if x > 0:
    print("positif")
elif x == 0:
    print("nul")
else:
    print("négatif")
```

Boucles

```
for i in range(3):
    print(i) # 0, 1, 2
```

```
while x > 0:
    x -= 1
```

9

9

break, continue et pass

```
for i in range(5):
    if i == 2:
        continue # saute le tour
    if i == 4:
        break # arrête la boucle
    print(i)

def f():
    pass # code à compléter plus tard
```

Fonctions utilitaires

- range(start, stop, step)
- enumerate(iterable) : indices + valeurs
- zip(a, b) : regroupe les éléments deux à deux

10

10

Définition simple

```
def f(x, y=1):  
    """Calcule la somme de deux nombres."""  
    return x + y
```

Annotation de type

```
def aire_rectangle(L: float, l: float = 1.0) -> float:  
    """Calcule l'aire d'un rectangle."""  
    if L < 0 or l < 0:  
        raise ValueError("Dimensions négatives interdites")  
    return L * l
```

Remarque : toute fonction devrait être commentée avec un docstring

11

11

Arguments spéciaux

- `*args` : nombre variable d'arguments positionnels
- `**kwargs` : arguments nommés

```
def exemple(a, *args, **kwargs):  
    print(a, args, kwargs)  
  
exemple(1, 2, 3, 'une chaîne', option1=True, option2=False)  
# 1 (2, 3, 'une chaîne') {'option1': True, 'option2': False}
```

12

12



La gestion des exceptions permet de traiter les cas imprévus.

```
try:
    x = int("abc")
except ValueError as e:
    print("Erreur : ", e)
else:
    print("Pas d'erreur")
finally:
    print("Bloc exécuté quoi qu'il arrive")
```

Pour générer une exception :

```
def division(a, b):
    if b == 0:
        raise ZeroDivisionError("Division par zero interdite")
    return a / b
```



13



Un **module** est un fichier Python (.py) contenant du code (fonctions, classes, variables).
Il permet de réutiliser du code dans plusieurs projets.

Exemple : math

```
import math

print(math.sqrt(16)) # 4.0
print(math.pi) # 3.141592653589793
print(dir(math)) # liste des fonctions / constantes disponibles
```



14

Problème

Comment faire si l'on veut juste utiliser 2 fonctions spécifiques d'un module ou que le nom du module est trop long?

Solutions :

- On peut importer seulement ce dont on a besoin
- Un **alias** raccourcit l'écriture

```
from math import sin, cos # Import
import numpy as np # Alias

print(cos(0)) # 1.0
print(np.arange(5)) # [0 1 2 3 4]
```
