Guilhem Saurel
gsaurel@laas.fr

Conception orientée Objet
I

Diapo dispo sur gsaurel site du LAAS homepages.laas.fr/gsaurel/talk/concept

↳ revoir Git et les forges github et gitlab.
↳ consulter documentation doxygen, sphinx.
↳ formattage automatique (black, clang-format, etc...)
↳ analyse statique (flake8, SonarQube, clang-tidy, etc...)
↳ test unitaires (catch tests, Boost UTF, gtest, pytest)
↳ intégration continue (gitlab-ci, github actions, travis, etc...)

Sommaire :
- Bloc 1 : C++ et Python
- Bloc 2 : ROS et Temps Réel
- + modules actifs

Evaluation :
- (15%) Bloc 1 : CR après 7h de TP (Python)
- (15%) Bloc 2 : CR après 7h de TP (Python et C++)
- (30%) contrôle Terminal 2h

I. Bases et introduction

1/ Affichage "Hello World" en C++

```
#include <iostream>
auto main() -> int {
    std::cout << "Hello\n";
    return 0;
}
```

nouvelle syntaxe pour commencer une fonction

↑ objet de la sortie par défaut (écran)

user / système

Compile avec le compilateur g++ (syntaxe shell) ⇒ utiliser powershell sous Windows

```
$ g++ hello.cpp && ./a.out
```

↳ exécute la commande de droite si la commande de gauche réussie

Résultat : hello

Astuce : vim nomfichier ouvre un éditeur dans le terminal

2/ Affichage "Hello World" en Python

```
#!/usr/bin/env python
if __name__ == "__main__":
    print("Hello")
```

N.B : fonctionne avec un simple print("Hello") mais if __name__ permet de compartimenter

Compile avec chmod +x hello.py

./hello.py

3/ Type de données en C++

auto ga {3}; ga est un int (4 bits de mémoire)

↑ le compilateur choisit le type pour optimiser

auto bu {3.14}; bu est un double (8 bits de mémoire)

const auto * const zo {"tau"}; est un char * const

std::string meu {"pi"}; gère la complexité des chaînes de caractères

Remarque : const indique que la variable est inchangée tout au long de la vie du programme.

4/ Type de données en Python

ga : int = 3

bu : float = 3,14

zo : str = "pi"

5/ Control flow : modifie l'ordonnancement d'éxécution ligne par ligne

5.a : En C++ on a ici une fonction qui ajoute 2 nombres

auto add (int first, int second) -> int { même logique de déclaration moderne

return first + second;

}

5.b En Python : on a ici la même fonction

def add(first: int, second: int) -> int:

return first + second

5. c)! Condition while en Python

user_input: int = 0

while user_input != 42:

user_input = int(input("guess : "))

transtypage de la saisie clavier

affichage avant la saisie

⚠ opérateur walrus : mais attention à la lisibilité

```
while (user_input := int(input("guess : "))) != 42:
    print("it's not", user_input)
```

5. d Break en C++

```
auto user_input {0};
while (true) {
    std::cout << "guess : ";
    std::cin >> user_input;
    if (user_input == 42) {
        std::cout << "yes!" << "\n";
        break;
    }
    std::cout << "It's not"
}
```

cf diapo

5. e Boucle for en C++

```
#include <cstdlib>
for (auto i{0}; i<1000; i++) {
    std::cout << "iteration" << i << "\n";
```

cf diapo

5. f Boucle for en Python

```
from random import randint

for i in range(10000):
    print("iteration", i)
    if randint() > 0.95:
        break
```

cf diap

5. g. Continue en C++

```
for (auto i{0}; i<10; ++i) {
    if (i%2 == 0) {
        continue;
    }
    std::cout << "iteration" << i << "\n";
}
```

continue ← reprend l'exécution au début de la boucle for

6/ Containers

En C++ : `using colors = std::vector<std::string>;` déclaration d'un tab de string

`colors cdors {"orange", "blue", "pink"};` initialisation du vecteur

`for (const auto &color : cdors) {` parcours de la liste par référence

prend successivement les valeurs

`std::cout << color << "\n";`

`}`

En Python

```
colors = ["orange", "blue", "pink"]
for color in colors:
    print(color)
```

NB: Tous les éléments de la liste n'ont pas besoin d'être du même type

7/ Objet

En C++

```
class Robot {
public:
    auto work() { battery -= 5; }
    auto get_battery() const -> int { return battery; }
protected:
    int battery{100};
};

auto main() -> int {
    auto robot = Robot{};
    std::cout << robot.get_battery() << "remaining \n";
    robot.work()
    std::cout << robot.get_battery() << "% remaining \n";
    return 0;
}
```

Les types sont OBLIGATOIRES pour les attributs !

↑ l'accès par l'utilisateur ne modifiera pas l'objet

instanciation

objet . méthode

En Python:

```
class Robot:
    battery = 100
    def work(self)
        self.battery -= 5
    def get_battery(self) -> int
        return self.battery

if __name__ == "__main__":
    robot = Robot()
    print(robot.get_battery(), "% remaining")
    robot.work()
    print(robot.get_battery(), "% remaining")
```

8_ Héritage:

```
class LeggedRobot : public Robot {
public:
    auto walk() { battery -= 10; }
};

auto main
```

cf. diapo

```
class LeggedRobo(Robot):
    def walk(self):
        self.battery -= 10

if __name__ == "__main__":
    robot = LeggedRobot()
    print(robot.get_battery(), "% remaining")
    robot.work()
    robot.walk()
```

Introduction à Git

01/09/2022

Créer un compte Git + créer une paire de clé cryptographique

+) New Repository : pour créer un projet

Owner = propriétaire du projet / Repository name = nom du projet (sans accents et espaces)
Organisation

Ajouter un README pour décrire le projet au format Markdown.

Choisir une licence BSD 2-Clause

Un projet vide contient : -> une licence
-> un README

Commit (modification des fichiers ou de l'arborescence)

↳ conseil : expliquer en quelques lignes la raison du commit

-> Browse permet de voir l'état du projet au moment du commit.

Fichier : Séparer les headers (contenant les prototypes publics) .hpp et la description fonctions.cpp dans 2 fichiers distincts.

Remarque :

```
#ifndef CONCEPTION_ORIENTEE_OBJET_EXAMPLE_ADDER_HPP
#define CONCEPTION_ORIENTEE_OBJET_EXAMPLE_ADDER_HPP
// déclaration des prototypes
#endif
```

Permet de s'assurer que le compilateur n'inclut qu'une seule fois la bibliothèque en tête du fichier

Fichier CMake : CMakeLists.txt permet de gérer la compilation de tous les fichiers dans leur dernière version, dans le dossier src