Chapitre 12. Autres langages, modules et bibliothèques (1/2)

Table des matières

1. Les langages de programmation

- 1.1 Origines
- 1.2 De nombreux langages
- 1.3 Langages de programmation
  - 1.3.1 Style
  - 1.3.2 Langages C et C++
  - 1.3.3 Comparaisons des langages

2. Modules et bibliothèques

- 2.1 Faire un module "mesfonctions"
- 2.2 Module math
- 2.3 Module random
- 2.4 Module Turtle
- 2.5 Bibliothèque Matplotlib

Remplissez le jupyter notebook suivant en vous aidant de votre livre de Première NSI de Serge BAYS .

- Pour répondre, double-cliquez sur **Réponse** et complétez la zone en-dessous. Puis cliquez sur le bouton *Exécuter*.
- **Important : pour fermer votre jupyter notebook, cliquez sur :**

Fichier / Créer une nouvelle sauvegarde

puis sur :

Fichier / Fermer et Arrêter

- Ecrivez ci-dessous votre prénom et votre nom :

Réponse :

Chapitre 12. Autres langages, modules et bibliothèques

1. Les langages de programmation

1.1 Origines

Lisez le paragraphe **Origines** p. 45

1) Quels sont les deux mathématiciens qui ont développés les premières machines à calculer ?

Réponse :

2) Qu'a utilisé Joseph-Marie Jacquard pour ses métiers à tisser ?

Réponse :

3) Qui a pensé le premier au principe de l'ordinateur ?

Réponse :

4) De quel ordinateur s'agissait-il ?

Réponse :

5) Qu'est-ce que concevoir un programme ?

Réponse :

6) Pour que les instructions du "langage machine" soient compréhensibles qu'a-t-on utilisé ?

Réponse :

7) A quoi correspondent les abréviations ADD et STR ?

Réponse :

8) Quels sont les deux langages de bas niveau ?

Réponse :

9) Qu'est ce que la sémantique telle que décrite au bas de la page 45 ?

Réponse :

1.2 De nombreux langages

Lisez le paragraphe **De nombreux langages** p. 46 au bas de la p. 47

10) Qu'est ce qui est le fondement de l'informatique ?

Réponse :

11) Citer un langage de haut niveau des années 1950 ? Pour quoi était-il utilisé ?

Réponse :

12) Citer un langage de haut niveau des années 1960 ? Pour quoi était-il utilisé (rechercher sur wikipédia)?

Réponse :

13) Qu'a permis le langage PASCAL ? A qui était-il destiné ?

Réponse :

14) Dans quel but a été créé le langage C en 1972 ?

Réponse :

15) Le langage C est la base de quels langages actuels ?

Réponse :

16) Qu'ont de particulier ces trois langages ?

Réponse :

17) Qu'est-ce que la programmation orientée objet ?

Réponse :

18) Quel système d'exploitation utilise java ?

Réponse :

19) Citer deux langages de programmation du web.

Réponse :

20) Citer un langage qui ne soit pas un langage de programmation

Réponse :

21) Pourquoi n'est-ce pas un langage de programmation ?

Réponse :

22) Que permet une requête SQL ?

Réponse :

23) Quels langages sont utilisés pour mettre en forme des textes mathématiques et des documents techniques ?

Réponse :

- Voici une belle formule mathématique. Pour voir son expression écrite en langage LaTeX, double cliquez dans la cellule ci-dessous :

$$\int_0^1 a_{-1}^2 \varphi_{-1}^2(x)\mathrm{d}x + \sum_{n=0}^{N} \sum_{k=0}^{2^n-1} \int_0^1 a_{n,k}^2 \varphi_{n,k}^2(x)\mathrm{d}x$$

1.3 Langages de programmation

1.3.1 Style

Lisez le paragraphe **Style** du bas de la p. 47 au haut de la p. 49

Voici le lien indiqué dans le livre :
https://fr.wikipedia.org/wiki/Comparaison_des_langages_de_programmation_multi-paradigmes
(https://fr.wikipedia.org/wiki/Comparaison_des_langages_de_programmation_multi-paradigmes)

Vous trouverez à cette adresse la définition d'un paradigme en informatique :
https://fr.wikipedia.org/wiki/Paradigme#Paradigme_en_informatique
(https://fr.wikipedia.org/wiki/Paradigme#Paradigme_en_informatique)

24) Qu'est-ce qu'un style impératif ?

Réponse :

25) Quels sont les principales constructions de ce style ?

Réponse :

26) Que permet une fonction ?

Réponse :

27) Qu'est-ce qu'un appel récursif ?

Réponse :

Voici un programme écrit en langage fonctionnel

In [ ]:
```
def u(n):
    if n > 1:
        calc = 2*u(n-1)     # Appel de la fonction mais pour la valeu
r n-1
    else:
        calc = 3            # Valeur du premier terme
    return calc
```

28) Calculer u(1), u(2), u(3), u(4) et u(5) et les afficher sur une seule ligne séparés par des ";"

In [ ]:
```

```

29) Ecrire un programme en langage impératif calculant et affichant les 5 premières valeurs de la question 28

In [ ]:
```
#  Ecrivez votre code ici :
n = 5


print(u)
```

30) En mathématiques, comment s'appelle cette suite de nombres ?

Réponse :

1.3.2 Langages C et C++

Lisez le paragraphe **langages C et C++** du milieu de la p. 49 au haut de la p. 52

31) Un programme en C peut-il être exécuté directement ?

Réponse :

32) Que permet le compilateur ?

Réponse :

Pour la compilation, voyez plus bas comment faire (rappel de la séquence 5)

33) Quelle est la fonction obligatoire dans un programme C ou C++ ?

Réponse :

34) Par quels symboles sont encadrés les commentaires en C ? et en C++ (voir séquence 5) ?

Réponse :

35) Par quoi est encadré le corps de la fonction et par quoi se termine chaque instruction ?

Réponse :

36) Comment déclarer les variables et où ?

Réponse :

Variables locales

Les variables déclarées à l'intérieur d'une fonction ou d'un bloc sont appelées variables locales.

Elles ne peuvent être utilisées que par des instructions contenues dans cette fonction ou ce bloc de code. Les variables locales ne sont pas connues pour les fonctions en dehors de leurs portées.

Exemple 1 :

```
//  Variable locale dans un programme en langage C
//  Exemple 1

#include <stdio.h>  // Importation de la bibliothèque standard (std)
  // d'input output (io). On y trouve par exemple la méthode printf.

void fonction_1()
{
    int a = 2;  // Vous pouvez utiliser a et b entre
    int b = 5;  // ces accolades seulement.
    printf("a = %d\n", a); // printf permet l'affichage formaté.
    printf("b = %d\n", b);
}

void fonction_2()
{
    printf("a = %d\n", a); // ERREUR, fonction_2() ne connait
                // aucune variable a
}

int main()  //  main() est la fonction exécutée en premier.
{
    printf("Exemple 1\n");
    fonction_1();
    fonction_2();
    printf("bye\n");
    return 0;  // main() de type int renvoie 0 signifie '0 erreur'.
}
```

- Lorsqu'on compile ce code, le compilateur signale l'erreur :

```
variable_locale_1.cpp: In function ‘void fonction_2()’:
variable_locale_1.cpp:16:24: error: ‘a’ was not declared in this scope
printf("a = %d\n", a); // ERREUR, fonction_2() ne connait
                   ^
```

- Dans la fonction_2() a variable 'a' n'a pas été déclarée dans ce "scope" c'est à dire dans la portée de la fonction_2().

- a et b sont appelés **variables locales**. Elles sont disponibles uniquement dans la fonction dans

laquelle elles sont définies (dans ce cas fonction_1()).

- Si vous essayez d'utiliser ces variables en dehors de la fonction dans laquelle elles sont définies, vous obtiendrez une erreur.

- Un autre point important est que **les variables a et b n'existent que jusqu'à l'exécution de fonction_1()**. Dès que la fonction fonction_1() se termine les variables a et b sont détruites.

Variables globales

Les variables globales sont définies en dehors d'une fonction, généralement en haut du programme. Les variables globales conservent leurs valeurs tout au long de la vie de votre programme et sont accessibles dans n'importe quelle fonction définie dans le programme.
Si une nouvelle valeur est affectée dans une fonction alors cette nouvelle valeur est accessible ensuite dans n'importe quelle autre partie du programme.

Exemple 2 :

```
//  Variable globale programme en C
//  Exemple 2

#include <stdio.h>

int a; //  Déclaration d'une variable globale

void fonction_1()  //  void (vide) est le type retourné. Ici rien.
{
    printf("Fonction 1\n");  //  \n sert au retour à la ligne.
    a = 5;   //  Affectation d'une valeur à la variable globale.
    printf("a = %d\n",a);
}

void fonction_2()
{
    printf("Fonction 2\n"); //  printf signifie impression formatée.
    printf("a = %d\n", a); //  %d représente les caractères qui servent
                           // à écrire l'entier 'a' en décimal.
}

int main(void)  //La fonction main() est du type int car elle retourne 0
//pour dire au système d'exploitation 'sortie sans erreur du programme'.
{
```

```
{
    printf("Exemple 2\n");
    fonction_1();
    fonction_2();
    printf("bye\n");
    return 0;
}
```

Voici ce qui s'affiche

```
Exemple 2
Fonction 1
a = 5
Fonction 2
a = 5
bye
```

Variable locale qui a le même nom qu'une variable globale

Les variables globales sont définies en dehors d'une fonction, généralement en haut du programme. Les variables globales conservent leurs valeurs tout au long de la vie de votre programme et sont accessibles dans n'importe quelle fonction définie dans le programme.

Exemple 2b :

```
//  Variable locale définie avec le même nom qu'une variable globale.
//  Exemple 2b

#include <stdio.h>

int a; //  Définition d'une variable globale

void fonction_1()  //  void (vide) est le type retourné. Ici rien.
{
    printf("Fonction 1\n");  //  \n sert au retour à la ligne.
    a = 5;   //  Affectation d'une valeur à la variable globale.
    printf("a = %d\n",a);
}

void fonction_2()
{
```

```
    printf("Fonction 2\n"); //  print f signifie impression formatée.
    printf("a = %d\n", a); //  %d représente les caractères qui servent
                           // à écrire l'entier 'a' en décimal.
}

void fonction_3()
{
    int(a)=144;  //  Déclaration localement de la variable 'a'.
    printf("Fonction 3\n");
    printf("a = %d\n", a);
}

void fonction_4()
{
    printf("Fonction 4\n");
    printf("a = %d\n", a);
}

int main(void)
{
    printf("Exemple 2b\n");
    fonction_1();
    fonction_2();
    fonction_3();
    fonction_4();
    printf("bye\n");
    return 0;
}
```

Voici ce qui s'affiche

```
Exemple 2b
Fonction 1
a = 5
Fonction 2
a = 5
Fonction 3
a = 144
Fonction 4
a = 5
bye
```

- Une variable locale peut être déclarée dans une fonction avec le même nom qu'une variable globale. Dans ce cas la valeur de la variable locale aura la priorité dans cette fonction.

- La valeur de la variable délarée localement est perdue en dehors de la fonction au profit de la valeur de la variable globale.

Paramètres formels

On appelle 'paramètre formel' un paramètre nommé dans les arguments d'une fonction. Les paramètres formels sont traités comme des variables locales au sein d'une fonction et ils ont la priorité sur les variables globales.

Exemple 3 :

```
//  Paramètres formels programme en C
//  Exemple 3

#include <stdio.h>

int a = 77; // Variable globale

void fonction_1(int a)  // nom_fonction(type-1 arg-1,...,type-n arg-n)
{
    a = 3;
    printf("Fonction 1\n");
    printf("a = %d\n",a);
}

void fonction_2()
{
    printf("Fonction 2\n");
    printf("a = %d\n", a);
}

int main(){
    printf("Exemple 3\n");
    fonction_1(a);  // L'argument 'a'  est appelé paramètre formel. Il
               // est traité dans la fonction comme une variable locale.
    fonction_2();
    printf("bye\n");
    return 0;
}
```

Voici ce qui s'affiche :

```
Exemple 3
Fonction 1
a = 3
Fonction 2
a = 77
bye
```

- On remarque que la variable globale 'a' n'a pas été modifiée par l'affectation d'une nouvelle valeur au paramètre formel 'a' à l'intérieur de la fonction_1.

Initialisation des variables locales et globales

Lorsqu'une variable locale est définie, elle n'est pas initialisée par le système, vous devez l'initialiser vous-même.
Les variables globales sont automatiquement initialisées par le système lorsque vous les définissez comme suit : Type de données Valeur par défaut initiale

int	0
char	'\0'
float	0
double	0
pointeur	NULL

C'est une bonne pratique de programmation d'initialiser les variables correctement.
Dans le cas contraire, votre programme pourrait produire des résultats inattendus, car les variables non initialisées prendront une valeur inattendue(garbage value) déjà disponible à leur emplacement mémoire.

Exemple de variables locales déclarées dans la fonction main()

Déclaration avec affection d'une valeur :

Exemple 4 :

```
int main()
{
    int a = 2; //  Déclaration de variables et affectation d'une valeur.
```

```
    char b = 'A';
    int valeur_ascii_de_b = int(b);
    float c = 13.59;
    double d = 1.7e104;
    int *pointeur_adresse_de_a;
    pointeur_adresse_de_a = &a;  // Le numéro de l'adresse mémoire de a.

    cout << "Exemple 4"  <<endl;

    cout << "Valeur de la variable a : ";
    cout << a << endl;   // cout << endl  affiche un saut de ligne.

    cout << "Valeur de la variable b : ";
    cout << b << endl;

    cout << "Valeur ASCII de b : ";
    cout << valeur_ascii_de_b << endl;

    cout << "Valeur de la variable c : ";
    cout << c << endl;

    cout << "Valeur de la variable d : ";
    cout << d << endl;

    cout << "Valeur de la variable adresse_de_a : ";
    cout << pointeur_adresse_de_a << endl;   // Ecriture hexadécimale.

    return 0;
}
```

Voici l'affichage :

```
Exemple 4
Valeur de la variable a : 2
Valeur de la variable b : A
Valeur ASCII de b : 65
Valeur de la variable c : 13.59
Valeur de la variable d : 1.7e+104
Valeur de la variable adresse_de_a : 0x7ed1de78
```

Exemple de variables locales déclarées dans la fonction main()

Déclaration sans affection d'une valeur :

```
int main()
{
    int e; //  Déclaration de variables sans affectation de valeur.
    char f;
    int valeur_ascii_de_f = int(f);
    float g;
    double h;
    int *p;


    cout << "Valeur de la variable e : ";
    cout << e << endl;

    cout << "Valeur de la variable f : ";
    cout << f << endl;


    cout << "Valeur ASCII de f : ";
    cout << valeur_ascii_de_f << endl;


    cout << "Valeur de la variable g : ";
    cout << g << endl;

    cout << "Valeur de la variable h : ";
    cout << h << endl;

    cout << "Valeur de la variable p : ";
    cout << p << endl;

    return 0;
}
```

Voici l'affichage :

```
Valeur de la variable e : 2127683228
Valeur de la variable f :
Valeur ASCII de f : 0
Valeur de la variable g : 1.4013e-45
Valeur de la variable h : 1.45263e-309
Valeur de la variable p : 0x20ef8
```

- On voit que dans le cas de variables locales déclarées sans affectation de valeur, *leurs valeurs sont aléatoires*.

Exemple de variables globales déclarées avant la fonction main()

Déclaration sans affection d'une valeur :

Exemple 4b :

```
//  Variable globale programme en C++
//  Exemple 4b

#include <iostream>  // Importe la bibliothèque iostream.
using namespace std;

int a; //  Déclaration de variables globales sans affectation de valeur.
char b;
int valeur_ascii_de_b = int(b);
float c;
double d;
int *pointeur_indefini;


int main()
{
    cout << "Exemple 4b"  <<endl;

    cout << "Valeur de la variable a : ";
    cout << a << endl;

    cout << "Valeur de la variable b : ";
    cout << b << endl;

    cout << "Valeur ASCII de b : ";
    cout << valeur_ascii_de_b << endl;

    cout << "Valeur de la variable c : ";
    cout << c << endl;

    cout << "Valeur de la variable d : ";
    cout << d << endl;
```

```
    cout << "Valeur du pointeur indéfini: ";
    cout << pointeur_indefini << endl;   // Ecriture hexadécimale.

    return 0;
}
```

Voici ce qui est affiché :

```
Exemple 4b
Valeur de la variable a : 0
Valeur de la variable b :
Valeur ASCII de b : 0
Valeur de la variable c : 0
Valeur de la variable d : 0
Valeur du pointeur indéfini: 0
```

- On remarque que toutes les valeurs des variables *globales* déclarées, tant qu'elles n'ont pas encore eu de valeur affectée, ont par défaut la valeur 0.
- Pour une variable de type caractère char, son code ASCII est 0 ce qui correspond au caractère Null. C'est pourquoi rien n'est affiché pour la valeur de la variable b.

37) Quelle sont les différences entre le C et le C++ ?

Réponse :

- Remarque sur

```
using namespace std;
```

On l'écrit au début d'un programme en C++ pour éviter de devoir écrire le préfixe ::std avant tous les mots clés de la bibliothèque standard du C++.

Ainsi :

```
using namespace std;

cout << "Hello world !" << endl;
```

```
...
```

évite de devoir écrire :

```
std::cout << "Hello world !" << std::endl;
```

38) Exemple de programme C++ et compilation

1. Si cela n'avait pas été fait au cours 5 "Entiers relatifs, réels et caractères" créez, par un clic droit, sur votre Raspberry, dans votre dossier "Documents" un sous dossier "langage_c".
2. Cliquez sur la framboise en haut à gauche, allez dans programmation puis ouvrez Geany (Geany est un éditeur de texte pour programmeur).
3. Dans Geany, allez dans le menu Fichier et choisissez Enregistrer sous... Déplacez-vous dans le dossier votre_username/Documents/langage_c et enregistrez ce nouveau fichier sous le nom mystere_c.cpp

Rappel : cpp est l'extension des fichiers sources (c'est à dire des fichiers écrits en langage compréhensible par des humains) en langage C++.

- Copiez coller dans Geany les lignes du programme en langage C++ ci-dessous.
  La différence entre le langage C et le langage C++ est que ce dernier est orienté objet.
  Par ailleurs, en langage C les commentaires s'écrivent entre les signes

  ```
  /* et */
  ```

tandis qu'en C++ on met simplement en début de ligne de commentaire le signe //.

Voici le programme :

```
#include <stdio.h>  //  Bibliothèque standard (std) input output (io)

int main()
{
   int p = 0, i, nbr;  //  Déclaration des variables.

   printf(" Entrez un nombre: ");
   scanf("%d", &nbr);  // scanf saisit les données entrées au clavier
```

```
                        // selon le format spécifié %d (entier décimal).
                        // &nbr est l'adresse mémoire de la variable nbr.

    for(i = 1; i <= nbr; i++)  // Ici i++ équivaut à i = i + 1
        {
        if(nbr%i == 0)
            {
            p++;  //  Ici p++ équivaut à p = p + 1
            }
        }

    if(p == 2)
        {
        printf(" %d en est un.\n",nbr);
        }
    else
        {
        printf(" %d n'en fait pas partie.\n",nbr);
        }

    return 0;
}
```

- Terminez en cliquant sur Fichier / Enregistrer.
- Quittez Geany.

La compilation

- Ouvrez un terminal et tapez la commande :

```
pwd
```

(pwd signifie Print Working Directory c'est à dire Afficher le Dossier Courant pour voir dans quel dossier vous êtes.

Placez-vous dans : /home/mon_username/Documents/langage_c
Pour vérifier la présence du fichier mystere.cpp, tapez la commande : ls

Vous devriez voir son nom.

On veut à présent compiler (c'est à dire traduire en langage machine) le fichier source mystere_c.cpp

- Saisissez la commande de compilation g++ (g++ fait partie de GNU Compiler Collection est un ensemble de compilateurs libres) suivante : sudo g++ mystere_c.cpp -o mystere_c

Exécution du programme mystere_c

- En passant par l'explorateur de fichiers (qui a pour icône un dossier jaune) vérifiez la présence dans le dossier Documents/langage_c de deux fichiers :

  L'un est le fichier source **mystere_c.cpp** écrit en C++ avec l'extension .cpp

  L'autre est le fichier exécutable **mystere_c** qui porte le même nom, mais sans extension.

- On va exécuter le fichier exécutable
- Pour cela saisissez dans le terminal la commande :

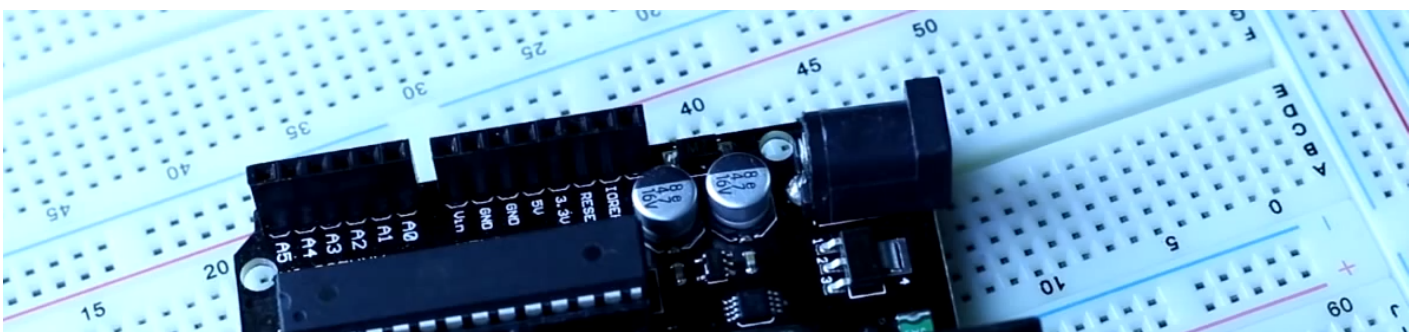
```
./mystere_c
```

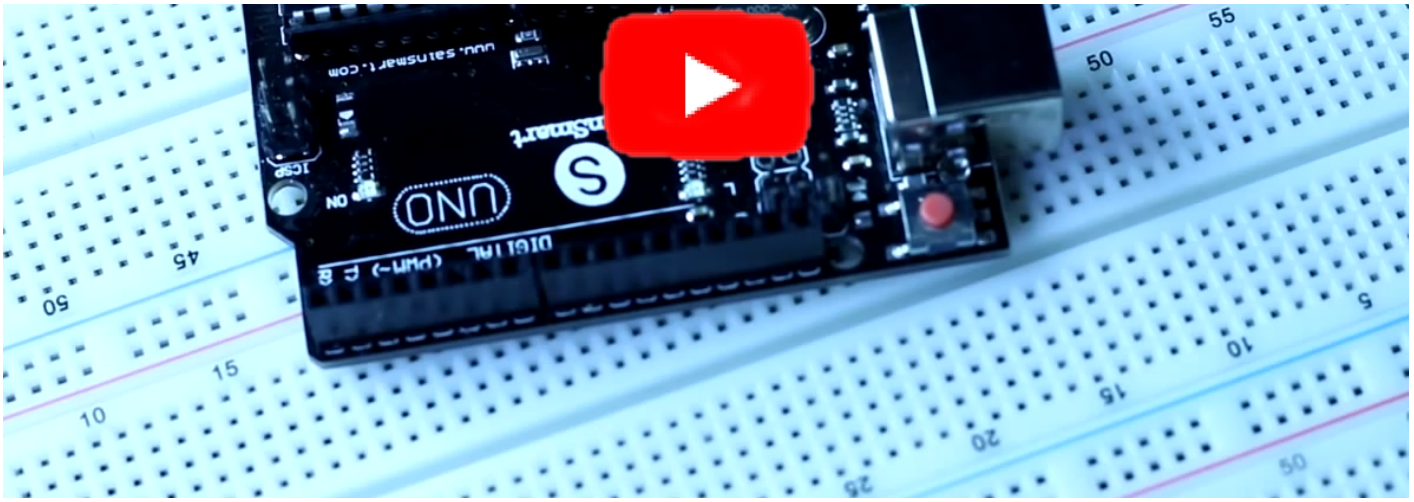
39) Que fait ce programme ?

Réponse :

EXERCICE : Travail sur un Arduino avec un simulateur

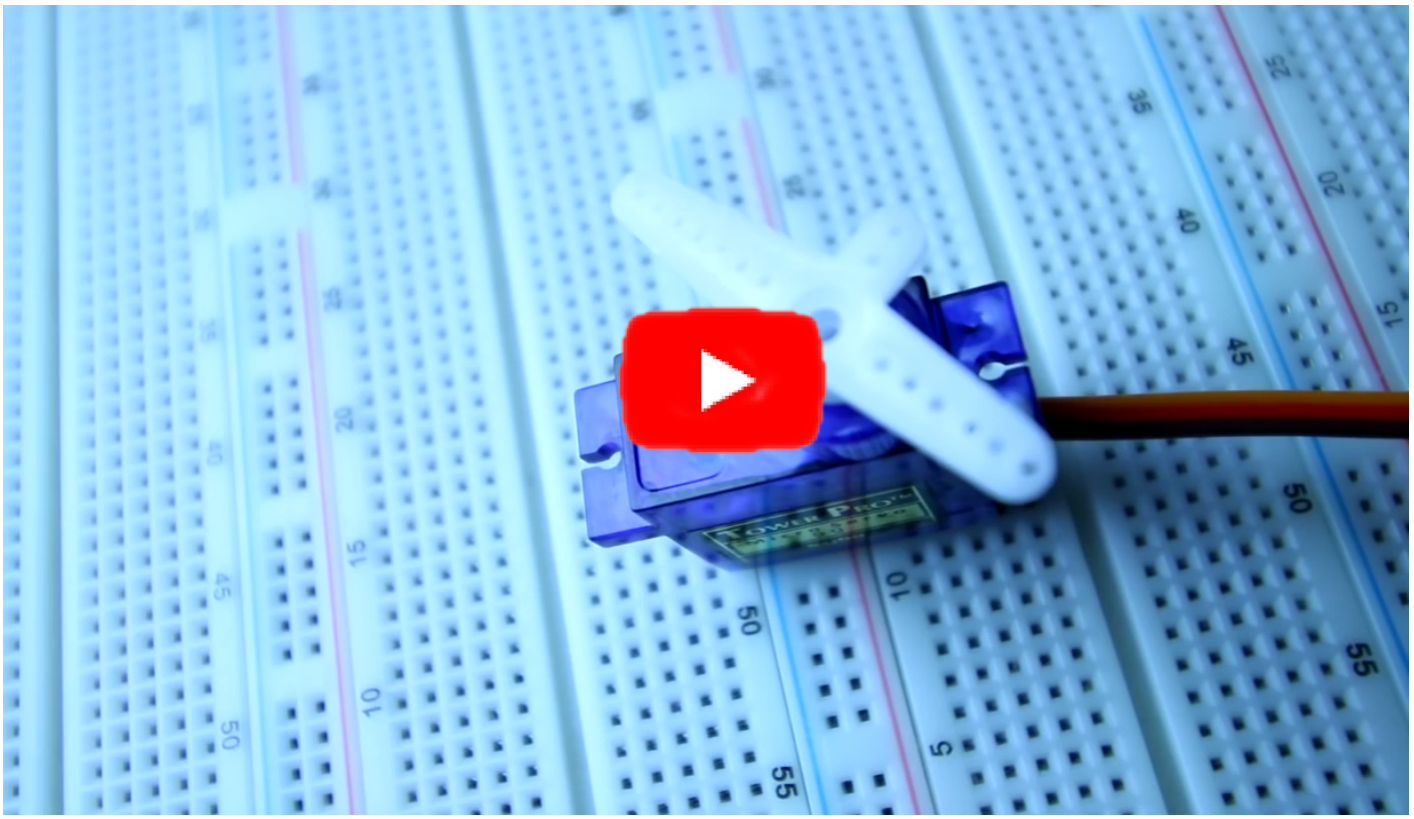
- Avant de commencer la simulation, regardez deux courtes vidéos de présentation de la carte Arduino réelle puis d'un servomoteur commandé par une carte Arduino.





(http://www.astrovirtuel.fr/jupyter/19_pnsi_cours/arduino.mp4)

Carte Arduino UNO



(http://www.astrovirtuel.fr/jupyter/19_pnsi_cours/arduino_servomoteur.mp4)

Servomoteur ou "moteur asservi"

- Nous voulons faire bouger un servomoteur (l'angle est compris entre 0 et 180°) en lui indiquant une vitesse (rapide (f) ou lente (l) et un nombre de pas à tourner (en degré).
- Pour faire bouger le servomoteur il faudra saisir des instructions du style : f30 , f-10, l5 , l-2 .....

- C'est vous qui saisirez ce qu'il y a à faire et ceci autant de fois que vous le voulez, la boucle étant infinie (il n'y a pas de fonction 'main' mais une fonction 'loop').
- Pour faire bouger le servomoteur de 10° rapidement, vous saisirez f10
- Pour faire bouger le servomoteur de -5° lentement, vous saisirez l-5

Au départ le servomoteur est placé à un angle de zéro degré.

Ce que vous devez faire :

- Compléter le programme écrit dans un langage proche du C et du C++.

Quelques explications :

Pour connaitre la position courante (en degrés entre 0 et 180), vous utiliserez :

```
pos_actuelle = myservo.read();
```

1) Pour le déplacement **rapide (fast)** (lettre f), vous utiliserez la fonction :

```
pos_finale = pos_actuelle + pas
myservo.write(pos_finale)
```

pos_finale étant la valeur de l'angle où aller.

2) Pour le déplacement **lent** (lettre l) :

- Il faudra déplacer le servomoteur de 1° en 1° (ou de -1° en -1°) avec un temps d'attente 'temps' en millisecondes entre chaque tour de boucle for.

```
delay(temps)
```

- Pour la boucle for :

voici un exemple :

```
for(int i = 0; i < pas; i += 1) // i += 1 équivaut à i = i + 1
    {
    ....
    }
```

Pour le if , voici un exemple :

```
if (pas > 0)
    {
    ......
```

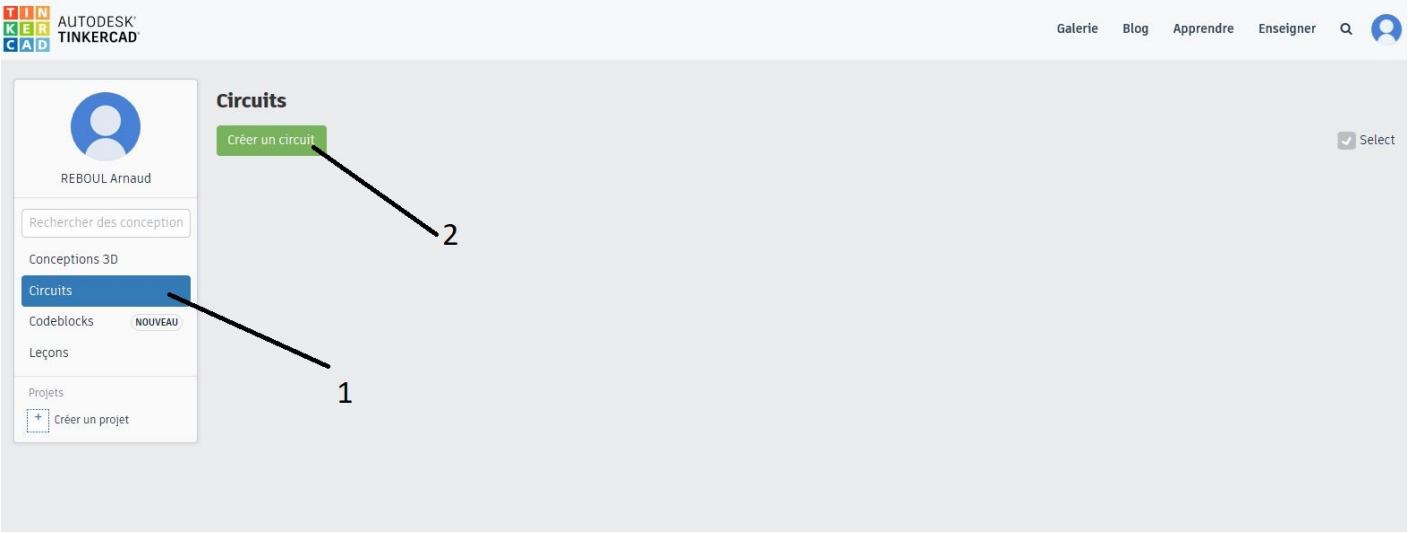
```
    }
else
    {
    .....
    }
```

Avant de programmer, vous devez d'abord créer votre circuit.

Voici le lien pour aller dans votre espace de travail :
https://www.tinkercad.com/joinclass/TVQ85WWK1A3Z
(https://www.tinkercad.com/joinclass/TVQ85WWK1A3Z)

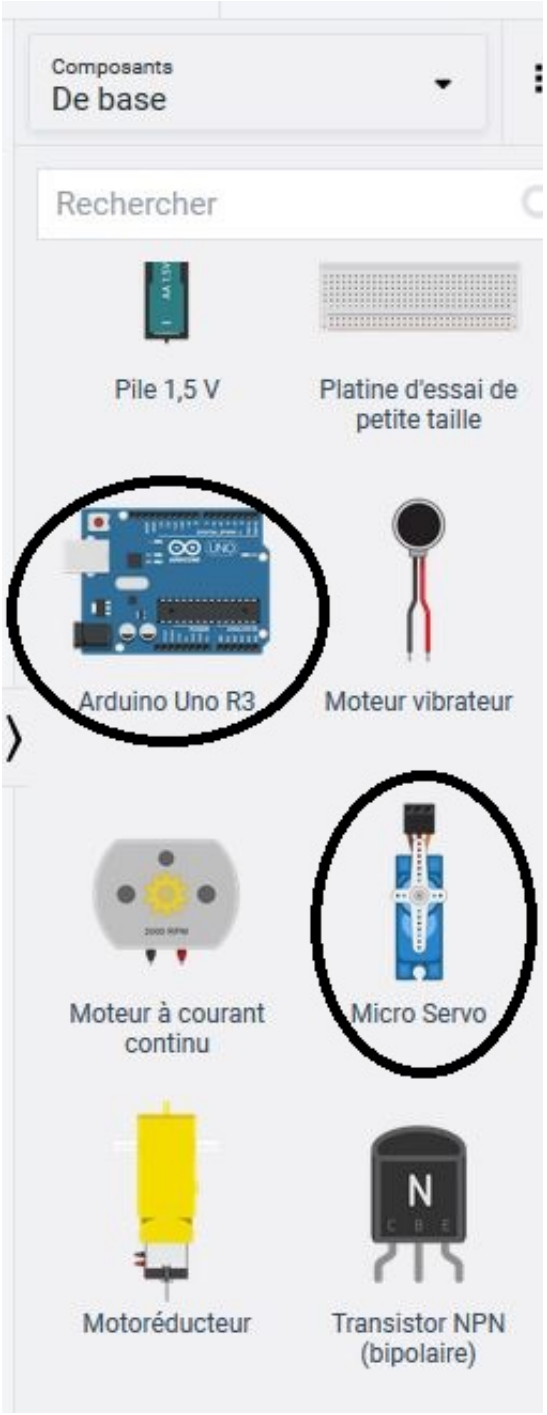
Votre pseudo c'est votre nom suivi de votre prénom sans espace (Pour Alain DUPONT le pseudo est **dupontalain**).

1) Vous arrivez dans votre salle.

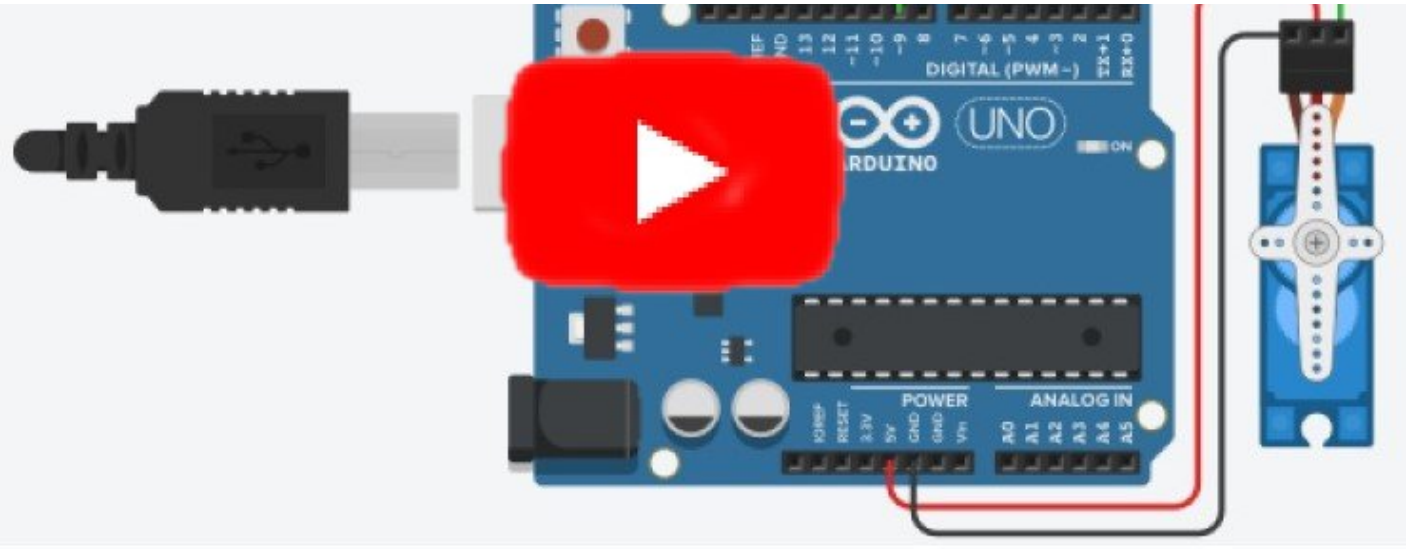


2) Création du montage

deux composants à choisir, cliquer dessus puis les placer

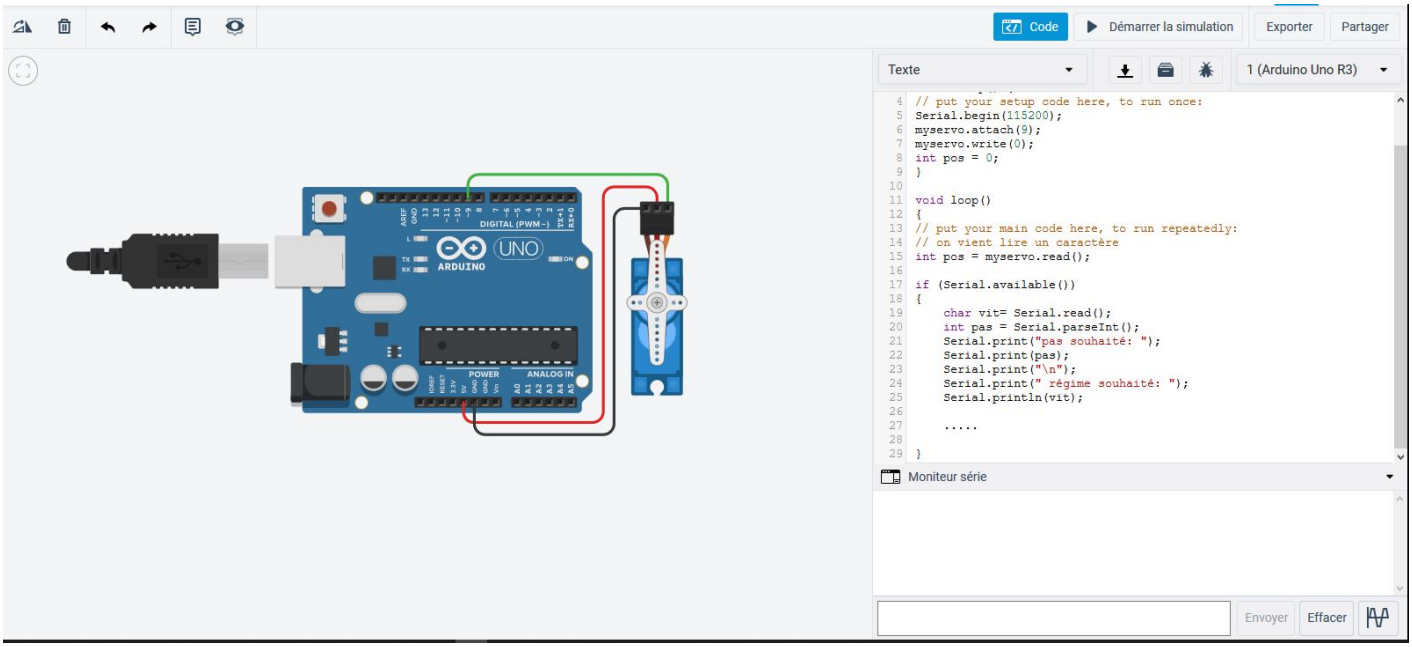


3) Branchement (explication en vidéo) Cliquez sur l'image ci-dessous.



(http://www.famille-reboul.fr/NSI/seq_12/explication_ardouino.mp4)

4) Voici ce que vous obtiendrez avec la recopie du code à droite



Soyez patient : le simulateur est très lent sur le Raspberry. Il faut attendre une vingtaine de secondes avant de le voir réagir après avoir envoyé une consigne dans le moniteur série.

Voici le code que vous devez copier et compléter.

```
#include <Servo.h>
Servo myservo;  // Création d'un objet Servo. On le nomme myservo.

// Fonction setup() pour initialiser
```

```
// Fonction setup() pour initialiser
void setup()
{
Serial.begin(115200);   // Communication série entre Arduino et ordi
myservo.attach(9);      // Attache le servo moteur au pin n°9
myservo.write(0);       // Place le servo moteur à 0°
int pos = 0;            // pos : angle où le servo moteur doit aller
}


void loop()   //  Devant le nom de la fonction 'loop', void signifie
              //  "vide" car rien n'est retourné par cette fonction.
{
int pos = myservo.read();  // La valeur de 'pos' est l'angle actuel.

// Pas de *main* ici, la boucle tourne indéfiniment:

// On vient lire la consigne donnée dans l'entrée série.

if (Serial.available())
    {
    char vit = Serial.read();  // On lit le 1er caractère (f ou l)
    int pas = Serial.parseInt();// On lit le nombre qui est après.
    Serial.print("Pas souhaite: ");
    Serial.print(pas);
    Serial.print("\n");               // Saut de ligne
    Serial.print("Regime souhaite: ");
    Serial.print(vit);
    Serial.print("\n\n");             // Double saut de ligne

    // Vos lignes de codes au dessous


    {
    ....
    }


    }
}
```

1.3.3 Comparaisons de langages

Lisez le paragraphe **Comparaisons** du haut de la p. 52 à la p. 57

40) Qu'est-ce qu'un langage interprété ?

40) Qu'est ce qu'un langage interprété ?

Réponse :

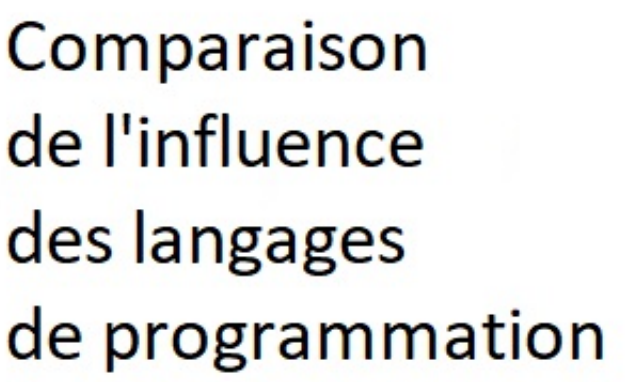
41) Donner le nom d'un langage interprété.

Réponse :

42) A quelle famille de langage appartiennent C et C++ ? Et qu'est ce que cela veut dire ?

Réponse :

Une dernière vidéo pour voir l'influence des langages de programmation



(http://www.famille-reboul.fr/NSI/seq_12/evolution_comparaison_des_langages.mp4)