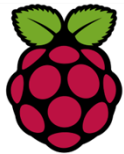




Atelier Raspi

Atelier N°2 Le courant électrique – la programmation

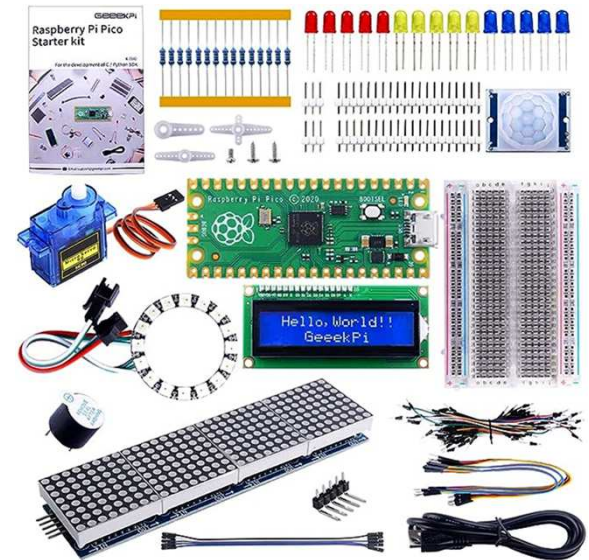


Logo du Raspberry Pico



Logo du MicroPython

L'atelier a pour valeurs, le partage, l'aide, la formation, le faire et construire ensemble à partir de l'expérience des participants



Atelier N°2 Le courant électrique – la programmation

Atelier N°2

Etape 1 Débit d'eau – Courant

Etape 2 le Multimètre

Etape 3 La résistance électrique, la LED

Etape 4 Le circuit électrique « LED avec pile »

Etape 5 Calcul de la résistance

Etape 6 Câblage des 4 leds

Etape 7 Programmation – Langage MicroPython

Etape 8 Les variables dans Python

Etape 9 Les types de variables et opérateurs standards

Etape 10 Les structures de contrôle 1

Etape 11 Les structures de contrôle 2

Etape 12 Les structures de contrôle 3

Etape 13 Listes et dictionnaires

Etape 14 Fonctions

Etape 15 Fonctions natives, fonction intégrée à python

Etape 16 Modules

Etape 17 Chenillard simple

Etape 18 Chenillard liste

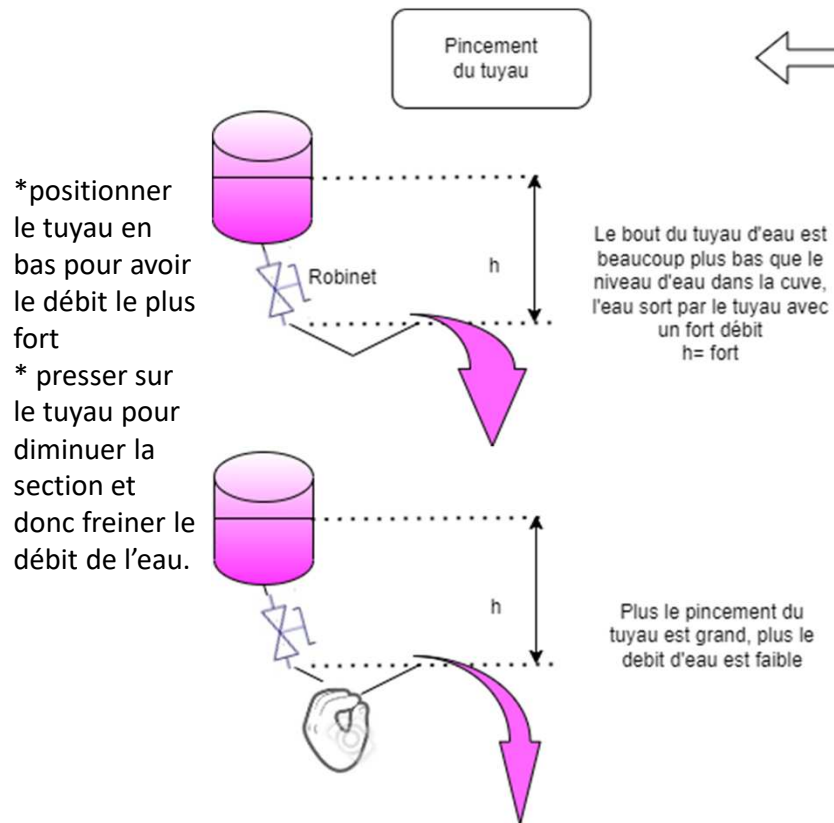
Etape 19 Chenillard évolué

Etape 20 Glossaire électronique

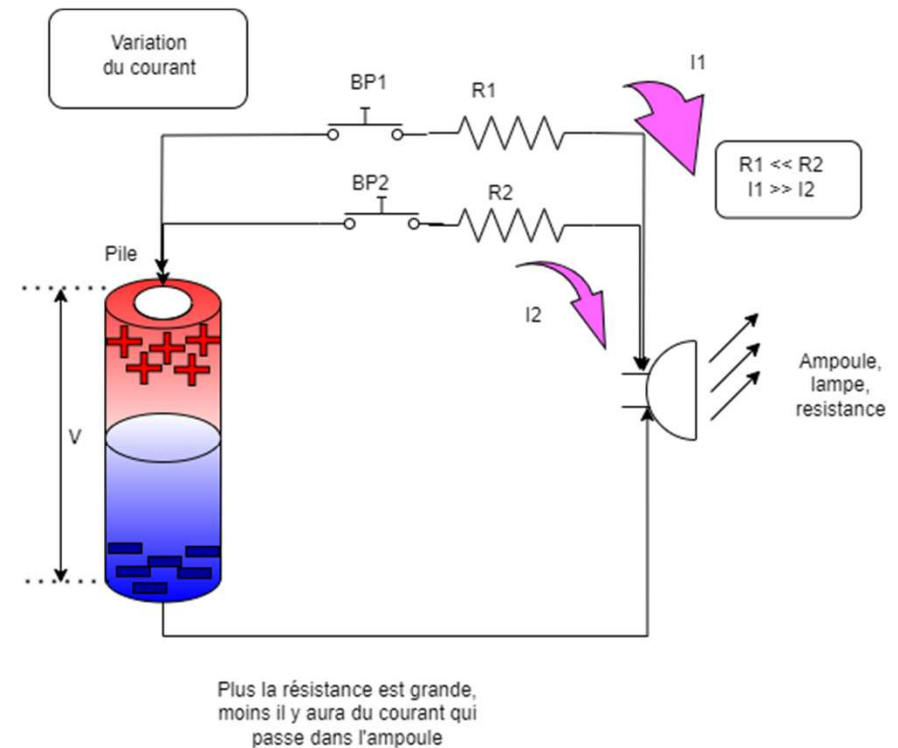
Etape 21 Glossaire informatique

A2E1 Débit d'eau – Courant

Retour sur la bouteille d'eau avec le tuyau



* On suppose que la cuve d'eau est alimentée en permanence



A2E2 Le multimètre

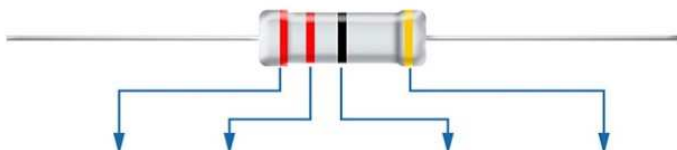


Cet appareil sert à mesurer différentes choses

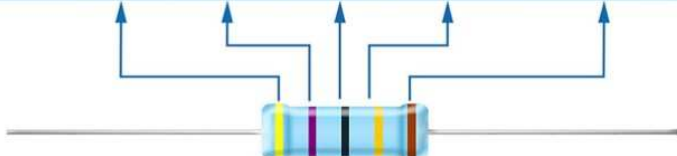
- La différence de potentiel entre deux points du circuit électrique
- Les résistances
- Les courants
- ...

Pour nos mesures, on va connecter le câble noir sur la borne COM et le câble rouge sur la borne Ω (En fonction des appareils)

A2E3 La résistance électrique, la LED



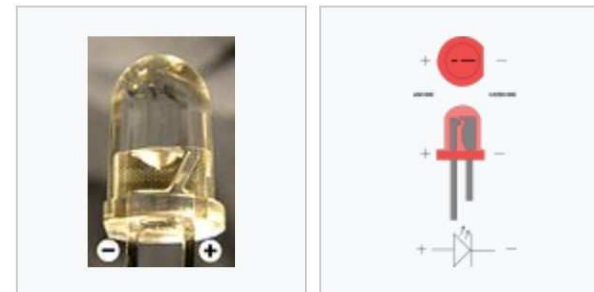
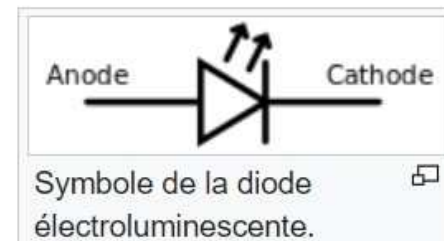
Color	1st Ring	2nd Ring	3rd Ring	4th Ring (Multiplier)	5th Ring (Tolerance)	
Black	0	0	0	1		
Brown	1	1	1	10	±1%	F
Red	2	2	2	100	±2%	G
Orange	3	3	3	1K		
Yellow	4	4	4	10K		
Green	5	5	5	100K	±0.5%	D
Blue	6	6	6	1M	±0.25%	C
Purple	7	7	7	10M	±0.1%	B
Gray	8	8	8		±0.05%	A
White	9	9	9			
Golden				0.1	±5%	J
Silver				0.01	±10%	K
No Color					±20%	M



E24 (± 5 %)
110, 120, 130, 150, 160, 180, 200, 220, 240, 270, 300, 330, 360, 390, 430, 470, 510, 560, 620, 680, 750, 820, 910, 1000

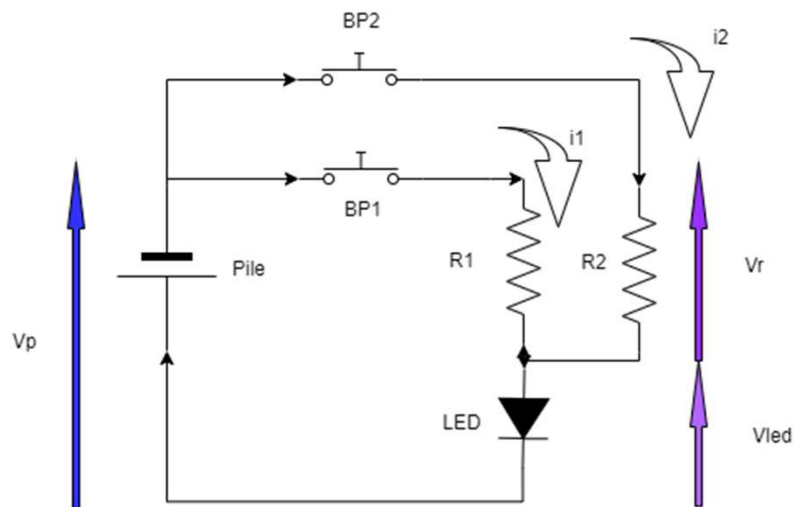
Mesure de la résistance électrique avec le contrôleur

- * fil de connexion, résistance=0
- * BP, résistance =0
- * résistance

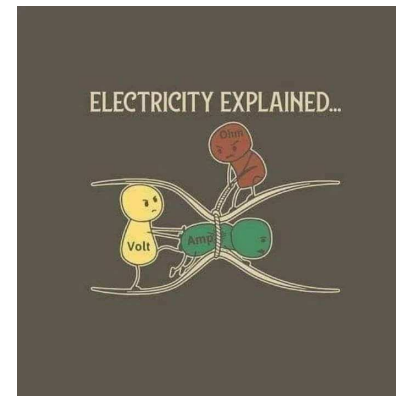


Le courant va du + vers le -

A2E4 Le circuit électrique « LED avec pile »



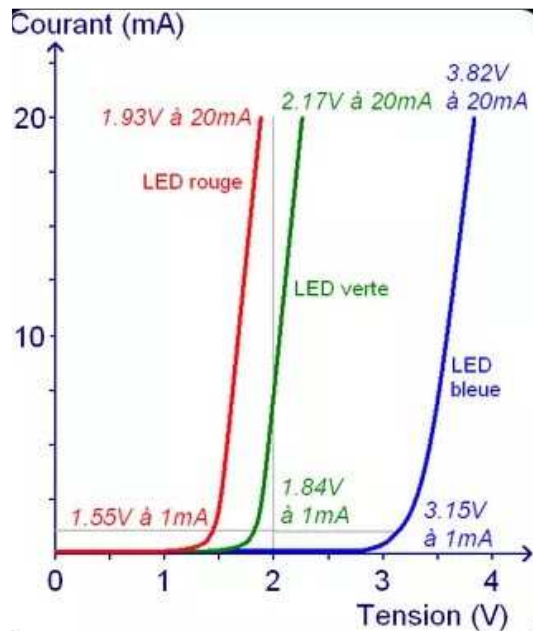
- La LED est plus brillante lorsque i est le plus grand.
- On va prendre $R2 = 1800 \text{ Ohms}$ $i2 = 0,72 \text{ mA}$ et $R1 = 222 \text{ Ohms}$, $i1 = 4,17 \text{ mA}$
- Plus la résistance est grande moins le courant passe et moins la LED brille.



suivant

Détail calcul
de R

A2E5 Détail du calcul de la résistance

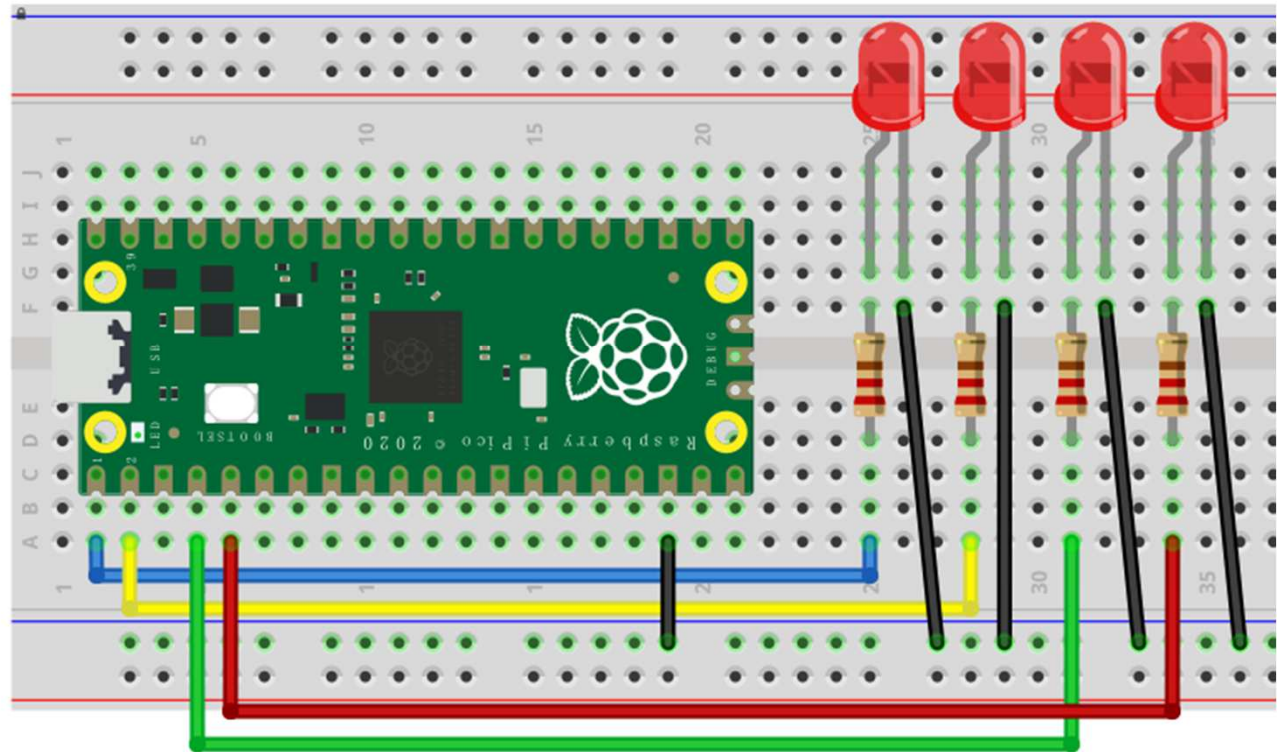


- Mesure de la tension V_{led} avec le voltmètre
- Comme le BP a une résistance = 0, $V_{pile} = V_{led} + V_r$ donc $V_r = V_{pile} - V_{led}$
- * Calcul de la résistance pour un courant donné
 $i = 1\text{mA}$ puis 20mA , $R = V_r / i$
- La LED est plus brillante lorsque i est le plus grand
- $V_{pile} = 3\text{V}$, pleine charge à $3,13\text{V}$
- Pour $i = 1\text{mA}$, V_{led} Rouge = $1,55\text{V}$ donc $R = (3,13 - 1,55) / 0,001 = 1580\text{ Ohms}$, pour $R1$, je vais prendre 1800 Ohms il passe alors $0,72\text{ mA}$. La tension batterie chute à $3,12\text{V}$
- Pour $i = 20\text{mA}$ V_{led} rouge = $1,93\text{V}$
Donc $R = (3,13 - 1,93) / 0,02 = 60\text{ Ohms}$.
- On va prendre 225 Ohms pour diminuer le courant (valeur de résistance du Kit), $i = 4,17\text{mA}$, $V_{led} = 1,91\text{V}$ et V_{batt} chute à $3,09\text{V}$

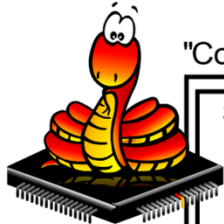
Avec le microcontrôleur la tension de sortie est de $3,3\text{V}$ et la résistance donnée dans le kit est de 225 Ohms valeur moyenne compte tenu du fait que l'on peut mettre des LEDs de différentes couleurs qui n'ont pas les mêmes caractéristiques.



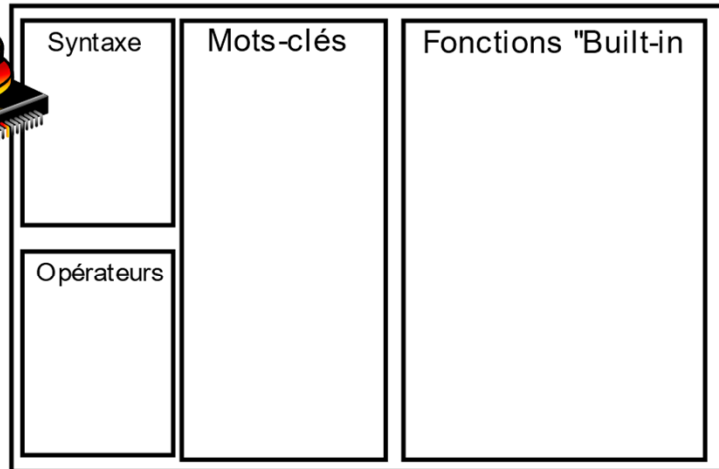
A2E6 Câblage des 4 leds



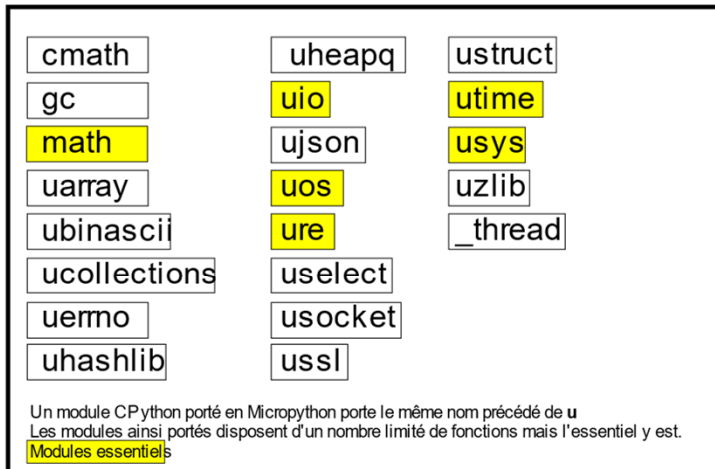
A2E7 Programmation – Langage MicroPython



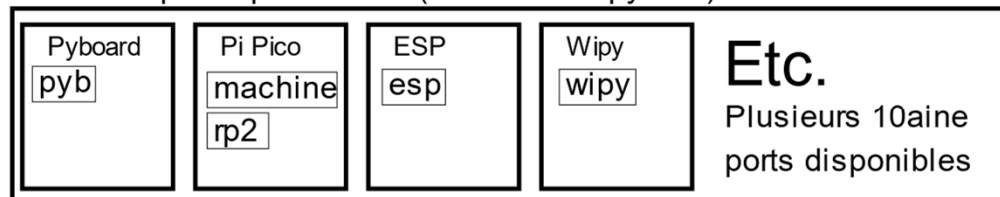
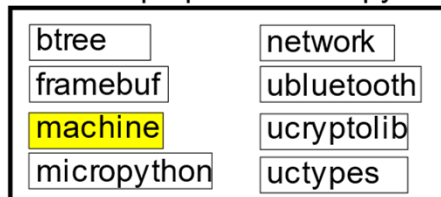
"Core" Micropython



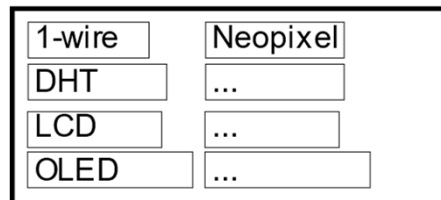
Modules standards



Modules propres à Micropython Modules spécifiques cartes ("Ports" Micropython)



Modules "bibliothèques" hardware



A2E8 Les variables

Une variable est un espace mémoire dans lequel on va stocker un objet.

C'est comme un tiroir ou l'on peut ranger (écrire), regarder (lire) ou enlever (supprimer) un objet.

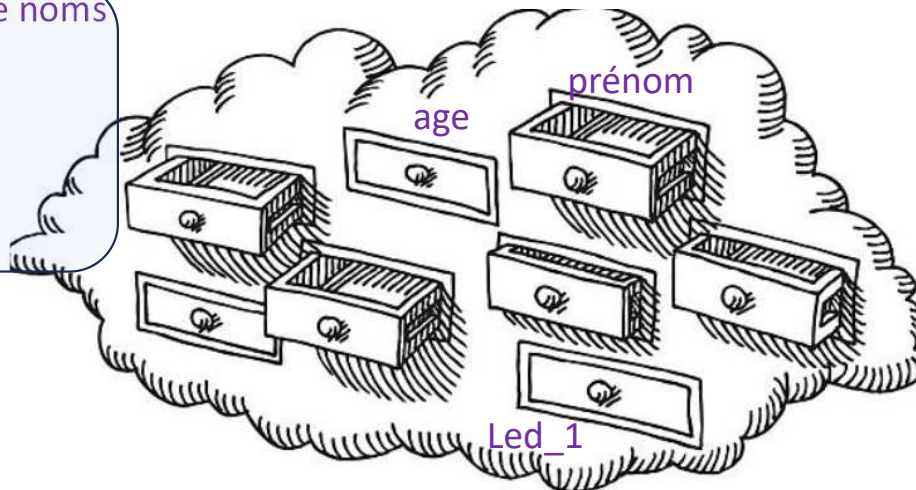
On va donner un nom à cette variable afin de pouvoir l'identifier, comme on met une étiquette sur un tiroir.

Ce nom doit suivre quelques règles :

- être compréhensible : décrire à quoi sert cette variable
- commencer par un caractère et non par un chiffre
- ne pas contenir de caractère spécial ni accentué
- ne pas contenir d'espace, si l'on souhaite donner un nom composé, utiliser le caractère souligné (underscore) '_'
- il faut rester cohérent dans la façon de nommer les variables

Exemples de noms de variable:

`age`
`led_1`
`led_2`



Affectation:

Pour écrire un objet dans une variable, on utilise l'opérateur '=' en mettant le nom de la variable à gauche de l'opérateur et l'objet à associé à droite.

Exemple:

`age = 10`

On peut lire le contenu de notre variable pour l'afficher:

`print(age)`

A2E9 Les types de variables et opérateurs standards

Une variable sans valeur (tiroir vide) est définie par : *age = None*

Type primitif:

- **bool** : définit un booléen qui peut prendre les valeurs **True** ou **False** (*Vrai ou faux*)
- **int** : définit un nombre entier (ex: 10)
- **float** : définit un nombre réel (ex :10,2)
- **str** : définit une chaîne de caractère (string, ex : « Bonjour tout le monde ! »)

Lors d'un test les valeurs suivantes sont considérée comme fausse : **False, None, "", [], (), {}**

Opération mathématique:

- **+** : addition, ex: **10+3** -> 13
- **-** : sustraction, ex: **10-3** -> 7
- ***** : multiplication, ex: **5*2** -> 10
- **/** : division, ex: **20/2** -> 10
- ****** : élévation a la puissance, ex: **6**4** -> 1296
- **//** : division entière, ex: **6//4** -> 1
- **%** : reste de la division entière, ex: **6%4** -> 2

Opérateurs de comparaison:

- **==** : égalité, **None==None** est vrai
- **!=** : inégalité, **None!=None** est faux
- **>, >=, <, <=** : comparaison, **6>3** est vrai
- **is, is not** : permet de comparer l'identité des objets

Opérateurs booléens:

- **and, or, not** : **not False** -> True

A2E10 Les structures de contrôle - 1

Indentation

En python l'indentation est obligatoire et permet de définir un bloc d'une ou plusieurs instructions, Une indentation est un retrait du code de 4 caractères, elle peut être insérée par 4 'espace' ou par la touche tabulation 'TAB'

Le symbole ':' à la fin d'une ligne d'instruction marque le début d'un bloc.

Seules les instructions suivantes permettent de définir le début d'un bloc:

- **def** : définition d'une fonction

Exemple:

```
def maFonction:  
    print('Bonjour')
```

A2E11 Les structures de contrôle - 2

Comparaison

Sert à comparer deux valeurs, le résultat d'une comparaison est un booléen qui prend la valeur **True** ou **False**.

< : strictement inférieur

<= : inférieur ou égal

== : égal

!= : différent

> : strictement supérieur

>= : supérieur ou égal

Exemple : **3 > 2 -> True**

L'instruction if, if...else, if...elif...else

L'instruction **if** Permet d'exécuter un bloc si la condition est vraie (**True**). Si ce n'est pas le cas on peut continuer le programme sans traiter le bloc d'instruction ou décider de traiter un autre bloc d'instruction délimité soit par **elif** si on souhaite rendre ce nouveau bloc conditionnel, soit par **else** si on souhaite le traiter par défaut.

Exemple:

n = -3

print(' le nombre {} est: '.format(n))

if n > 0:

print('positif')

elif n < 0:

print('négatif')

else :

print('zéro')

A2E12 Les structures de contrôle - 3

Boucle while

La boucle **while** exécute un bloc tant que la condition est vraie (**True**).

Exemple:

```
n = 3  
while n > 0:  
    print(n)  
    n = n - 1  
print('fini !')
```

Lorsque l'on écrit un programme pour un microcontrôleur, il y a toujours une boucle **while True**: que l'on appelle la boucle principale.

Boucle for

La boucle **for** permet de parcourir un ensemble de valeur une à une dans l'ordre de la première à la dernière.

Exemple:

```
for c in 'hello':  
    print(c)
```

la variable **c** va prendre tour à tour les valeurs 'h', 'e', 'l', 'l', 'o'

A2E13 Listes et dictionnaires

Listes

La liste est une séquence linéaire d'objet. Elle est délimitée par des `[]` et ses éléments sont séparés par des `,`. Une liste peut contenir plusieurs types d'objet. On peut utiliser le constructeur `list()` pour créer une liste vide,

Exemple:

```
N = [123, 25, 12]
```

```
B = ['chat', 12, 'chien']
```

La fonction `len` permet de connaître la longueur d'une liste (son nombre d'élément):

```
len(N) -> 3
```

On peut parcourir une liste par itération:

```
for item in N:
```

```
    print(item)
```

Ou par index:

```
for index in range(len(N)):
```

```
    print(N[index])
```

Dictionnaires

Comme les listes les dictionnaires peuvent contenir plusieurs objets de type différents. Ceux-ci seront toujours sous la forme d'une paire **key:valeur**. Le constructeur `dict()` permet de créer un dictionnaire vide. Un dictionnaire est délimité par `{}`.

Exemple:

```
N = {'voiture':'4 roues', 'moto':'2 roues'}
```

```
B = {'animal':'chat', 'taille_poids':[60,10]}
```

On accède à la valeur d'un **key** en l'utilisant comme index:

```
print(N[voiture]) -> '4 roues'
```

A2E14 Fonctions

Fonctions

Une fonction permet d'écrire un bloc de code que l'on va pouvoir appeler plusieurs fois à différent moment de notre programme sans avoir à le réécrire. Une fonction doit être définie avant de pouvoir être appelée. On définit une fonction à l'aide du mot clé ***def nom_de_ma_fonction(args)*** : on peut ensuite l'appeler par ***nom_de_ma_fonction()*** depuis notre programme principal ou depuis une autre fonction.

Une fonction peut prendre un ou plusieurs arguments en entrée et retourner un objet.

Exemple:

def saluer(nom):

print('bonjour', nom)

Utilisation de la fonction:

saluer('Philippe')

Saluer('marc')

Rendu du programme:

Bonjour Philippe

Bonjour marc

A2E15 Fonctions natives, fonction intégrée à python

abs	Retourne la valeur absolue d'un nombre
aiter	Renvoie un itérateur asynchrone
all	Retourne True si tous les éléments d'un itérable sont vrais
anext	Renvoie l'élément suivant d'un itérable asynchrone
any	Retourne True si au moins un élément d'un itérable est vrai
ascii	Retourne une string qui représente l'objet
bin	Converti un nombre en binaire sous forme de string
bool	Converti en élément en valeur booléenne
breakpoint	Place dans le débogueur
bytearray	Retourne un tableau de taille donnée
bytes	Retourne un objet de type "byte"
callable	Test si l'objet est "appelable"
chr	Converti un nombre en sa valeur ascii sous forme de string
classmethod	Retourne la méthode de classe pour une fonction
compile	Exécute puis retourne du code Python
complex	Retourne un nombre complexe
delattr	Supprime un attribut de l'objet
dict	Crée et retourne un dictionnaire {}
dir	Retourne une liste des attributs d'un objet Python
divmod	Retourne un tuple avec le quotient et le reste d'une division
enumerate	Retourne un objet enumerate
eval	Exécute du code Python
exec	Exécute du code dynamiquement
filter	Crée un itérateur à partir d'éléments qui renvoient True
float	Retourne un nombre à virgule flottante
format	Retourne une représentation formatée d'une valeur
frozenset	Retourne un objet immutable frozenset
getattr	Retourne la valeur de l'attribut nommé d'un objet
globals	Retourne un dictionnaire avec la table des symboles globaux
hasattr	Test si la l'objet a l'attribut ou pas
hash	Retourne un entier avec la valeur de hachage d'un objet
help	Appelle l'aide native de Python
hex	Converti un entier en nombre de base hexadécimale

id	Renvoie l'identification d'un objet
input	Lit et renvoie une ligne de chaînes de caractères
int	Converti un nombre ou une string en nombre entier
isinstance	Test si un objet est une instance de la classe
issubclass	Test si un objet est une instance d'une autre classe
iter	Retourne un itérateur
len	Retourne la longueur d'un objet
list	Crée une nouvelle liste ou transforme un autre objet en liste
locals	Retourne un dictionnaire avec la table des symboles locaux
map	Applique une fonction et retourne une liste
max	Retourne le plus grand élément
memoryview	Retourne une identification mémoire d'un objet de type byte
min	Retourne le plus petit élément
next	Retourne l'élément suivant d'un itérateur
object	Crée un objet Python
oct	Converti un entier en nombre de base 8
open	Retourne un objet de type fichier
ord	Retourne un entier d'un caractère avec sa valeur unicode
pow	Retourne la puissance d'un nombre
print	Permet d'afficher un objet sur la sortie standard
property	Retourne un objet "property"
range	Retourne une liste de nombres
repr	Retourne une représentation d'un objet
reversed	Retourne un itérateur dans le sens inverse
round	Retourne un nombre arrondi à une décimale donnée
set	Crée et retourne un set d'éléments uniques (sans doublon)
setattr	Définit la valeur d'un attribut de l'objet
slice	Coupe et retourne un objet
sorted	Trie et retourne un itérable
staticmethod	Transforme une méthode en une méthode statique
str	Retourne une chaîne de caractères
sum	Fait la somme des éléments d'un itérable
super	Retourne un objet proxy de la classe
tuple	Crée et retourne un tuple
type	Retourne le type d'un objet Python
vars	Retourne l'attribut __dict__
zip	Prend des itérables et retourne une liste de tuples
import	Fonction appelé par l'instruction import

A2E16 Modules

Modules

Un module contient un ou plusieurs fichier python que l'on peut importer en début de script et qui contient un ensemble de définitions. Exemple le module ***time*** contient des fonctions permettant de dormir.

Exemple:

```
import time
```

```
time.sleep(3) # dort pendant 3 secondes
```

Un module spécifique à l'utilisation du raspi pico existe, c'est lui qui va nous permettre d'interagir avec les périphériques du microcontrôleur et donc avec le monde réel : ***machine***

Exemple:

```
from machine import Pin
```

```
P0 = Pin(0, Pin.OUT)
```

```
P0.value(1)
```

A2E17 Exercice – chenillard simple

Objectif :

A partir du câblage des 4 leds réalisé plus tôt, effectuer la séquence suivante:

initialisation

Importer les modules nécessaires

Déclarer les 4 leds comme étant des I/O en sortie

Eteindre toutes les leds

Boucle infinie:

- inverser l'état de la led1

- attendre une seconde

- inverser l'état de la led2

- attendre une seconde

- inverser l'état de la led3

- attendre une seconde

- inverser l'état de la led3

- attendre une seconde

A2E18 Exercice – chenillard liste

Objectif :

A partir du câblage des 4 leds réalisée plus tôt, effectuer la séquence suivante:

initialisation

Importer les modules nécessaires

Déclarer les 4 leds comme étant des I/O en sortie

Déclarer une liste contenant les 4 leds dans l'ordre

Eteindre toutes les leds par itération sur les éléments de la liste

Boucle infinie:

en itérant sur les éléments de la liste:

inverser l'état de l'item

attendre une seconde

A2E19 Exercice – chenillard évolué

Objectif :

A partir de l'exercice précédent effectuer les modifications nécessaires sur la liste pour réaliser les chenillards suivants:

1/ led1:on, led2:on, led3:on, led4:on, led4:off, led3:off, led2:off, led1:off

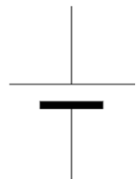
2/ led1:on, led1:off, led2:on, led2:off, led3:on, led3:off, led4:on, led4:off

3/ proposer votre séquence ...

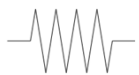
A2E20 Glossaire électronique



Symbole de la diode LED



Pile



Résistance



BP

A2E21 Glossaire informatique

Instruction du langage : ne nécessite pas d'importer de module pour les utiliser

print(« Bonjour ») : affiche Bonjour sur la console du PC

While test : Boucle tant que test est VRAI

import : pour importer des modules contenant des fonctions

import time : permet d'appeler les fonctions relatives au temps

time.sleep(x) : arrête l'exécution du programme pendant x secondes

import machine : module contenant les fonctions permettant d'agir sur les périphériques du pi pico

from : pour simplifier l'écriture on importe certaines fonctions directement dans une variable à l'aide de « from »

from machine import Pin : permet d'accéder aux fonctions du module machine.Pin plus facilement

ma_led_verte = Pin(n, Pin.OUT) : permet d'indiquer comment on souhaite utiliser un GPIO,
ici on a configuré le pin I/O N° n en sortie tout ou rien

ma_led_verte.Toggle() : inverse l'état de l'I/O associé à la variable.

Ici l'IO n passera de l'état haut (+3.3v) à l'état bas (0v) ou inversement

: En début de ligne permet d'écrire un commentaire, ce qui est écrit n'est pas une instruction et donc n'est pas pris en compte par le programme.

Variable $A=A+1$ -> Nouvelle valeur = ancienne valeur+1

print (A, " texte "), **input** ("texte", B)

Opérateurs: +,-,*,/,%, **Données**: int(), float(), str(), bool()=True or False