

CE

CONRAD

Sommaire

1	De l'installation du système d'exploitation au premier programme Python	5
1.1	De quoi avez-vous besoin ?	5
1.1.1	Chargeur pour téléphone portable à prise micro-USB	6
1.1.2	Carte mémoire	6
1.1.3	Clavier	6
1.1.4	Souris	6
1.1.5	Câble réseau	6
1.1.6	Câble HDMI	6
1.1.7	Câble audio	7
1.1.8	Câble vidéo composite jaune (FBAS)	7
1.2	Installer le système d'exploitation Raspbian	7
1.2.1	Préparer la carte mémoire sur l'ordinateur	7
1.2.2	Installateur de logiciel NOOBS	8
1.2.3	Les voyants LED sur le Raspberry Pi	8
1.2.4	Le premier démarrage sur le Raspberry Pi	9
1.3	Presque comme Windows - l'interface graphique LXDE	9
1.3.1	Enregistrer vos fichiers sur le Raspberry Pi	11
1.4.1	Deviner les chiffres avec Python	14
1.4.2	Voilà comment cela fonctionne	16
2	Le premier voyant LED s'allume sur le Raspberry Pi	18
2.1	Composants dans le kit	19
2.1.2	Câble de connexion	20
2.1.3	Résistances et leur code couleur	21
2.2	Brancher les LED	22
2.3	GPIO avec Python	26
2.4	Allumer et éteindre une LED	27
2.4.1	Voilà comment cela fonctionne	28
2.5	Lancer Python avec support GPIO sans terminal	29
3	Feu de signalisation	32
3.1.1	Voilà comment cela fonctionne	34
4	Feu pour piétons	36
4.1.1	Voilà comment cela fonctionne	38
4.2	Bouton sur la connexion GPIO	39
4.2.1	Voilà comment cela fonctionne	43
5	Motif à LED coloré et chenillard	45
5.1.1	Voilà comment cela fonctionne	48

6	Variation de l'intensité lumineuse des LED par modulation de la largeur des impulsions	53
6.1.1	Voilà comment cela fonctionne	56
6.1.2	Faire varier l'intensité lumineuse de deux LED indépendamment l'une de l'autre	57
6.1.3	Voilà comment cela fonctionne	59
7	Indicateur à LED montrant l'espace disponible des cartes mémoires	60
7.1.1	Voilà comment cela fonctionne	63
8	Dé graphique	65
8.1.1	Voilà comment cela fonctionne	67
9	Horloge analogique sur l'écran	72
9.1.1	Voilà comment cela fonctionne	73
10	Boîte de dialogue graphique pour contrôler le programme	77
10.1.1	Voilà comment cela fonctionne	79
10.2.1	Voilà comment cela fonctionne	84
10.3	Régler la vitesse de clignotement	87
10.3.1	Voilà comment cela fonctionne	88
11	Pitance avec les LED	90
11.1.1	Voilà comment cela fonctionne	93

1.1.1 Chargeur pour téléphone portable à prise micro-USB

Un chargeur moderne pour téléphone portable suffit pour faire fonctionner le Raspberry Pi. Les anciens chargeurs construits avec les premières technologies de recharge via USB ne sont pas assez puissants. Si vous utilisez un périphérique USB à forte consommation électrique comme un disque dur sans alimentation externe, vous devrez utiliser un bloc d'alimentation plus puissant. Le bloc d'alimentation doit délivrer une tension de 5 V et un courant d'au moins 700 mA, de préférence 1000 mA. Le régulateur de puissance intégré empêche que les composants ne « grillent » si vous utilisez un bloc d'alimentation trop puissant.

Un bloc d'alimentation trop faible se manifeste ainsi.

Lorsque le Raspberry Pi démarre deux fois mais que le pointeur de la souris ne se bouge pas ou que le système ne réagit pas aux commandes du clavier, cela indique que l'alimentation électrique n'est pas suffisante. De même, si vous ne pouvez pas accéder aux données de votre clé USB ou des disques durs, vous devrez utiliser un bloc d'alimentation plus puissant.

1.1.2 Carte mémoire

La carte mémoire sert de disque dur dans le Raspberry Pi. Elle contient le système d'exploitation. Les données internes et les programmes installés y sont également conservés. La carte mémoire doit être d'au moins 4 Go et prendre en charge la norme de classe 4 selon les indications du fabricant. Cette norme indique la vitesse de la carte mémoire. Une carte mémoire de classe 10 récente présente des performances remarquables.

1.1.3 Clavier

Tout clavier disposant d'une connexion USB peut être utilisé. Les claviers sans fil ne fonctionnent pas toujours car ils ont besoin de beaucoup d'énergie ou de pilotes spéciaux. Si vous n'avez pas d'autre clavier, vous devrez utiliser un concentrateur hub USB doté d'une alimentation électrique propre pour faire fonctionner le clavier sans fil.

1.1.4 Souris

Une souris avec connexion USB n'est nécessaire que si vous utilisez un système d'exploitation avec interface utilisateur graphique sur le Raspberry Pi. Certains claviers disposent de ports USB pour les souris de sorte que vous n'avez pas à fournir d'autres ports de connexion. Vous pouvez ensuite les utiliser pour connecter des clés USB p. ex.

1.1.5 Câble réseau

Pour vous connecter à un réseau local avec le routeur, un câble réseau est nécessaire. Il est nécessaire dans tous les cas pour la configuration initiale. Vous pourrez également utiliser ultérieurement une connexion WLAN. De nombreuses fonctions du Raspberry Pi ne sont pas utilisables sans accès à Internet.

1.1.6 Câble HDMI

Le Raspberry Pi peut être connecté à un écran/moniteur ou un téléviseur à l'aide d'un câble HDMI. Pour le connecter à un moniteur d'ordinateur avec une prise DVI, il existe des câbles HDMI ou adaptateurs spéciaux.

1.1.7 Câble audio

Un casque audio ou des enceintes pour ordinateur peuvent être utilisées sur le Raspberry Pi via un câble audio avec prises jack de 3,5 mm. Le signal audio est également disponible via le câble HDMI. Aucun câble audio n'est nécessaire pour les téléviseurs ou moniteurs HDMI. Si un moniteur d'ordinateur est connecté via un câble HDMI avec adaptateur DVI, le signal audio est en général perdu à ce point de sorte que vous avez besoin d'utiliser à nouveau la sortie audio analogique.

1.1.8 Câble vidéo composite jaune (FBAS)

Si vous n'avez pas de moniteur HDMI, le Raspberry Pi peut être connecté à un téléviseur classique en utilisant un câble vidéo composite analogique, avec les fiches jaunes caractéristiques. Cependant, la résolution de l'écran sera très faible. Pour les téléviseurs sans entrée composite jaune, il existe un adaptateur composite (FBAS) vers SCART. L'interface graphique peut être utilisée seulement avec des restrictions dans la résolution analogique du téléviseur.

1.2 Installer le système d'exploitation Raspbian

Le Raspberry Pi est livré sans système d'exploitation. Contrairement aux autres ordinateurs qui utilisent presque tous Windows, une version spécialement adaptée de Linux est recommandée pour le Raspberry Pi. Windows ne fonctionnerait pas avec des équipements économes en énergie.

La version de Linux, que le fabricant du Raspberry Pi recommande et prend en charge, s'appelle Raspbian. Raspbian repose sur la version Debian de Linux, une des versions les plus connues de Linux, qui est basée entre autres sur les versions populaires Ubuntu et Knoppix de Linux. Dans les ordinateurs classiques, il y a un disque dur. Dans Raspberry Pi, il y a une carte mémoire. Vous y trouverez le système d'exploitation et les données, à partir desquelles le Raspberry Pi va également démarrer.

1.2.1 Préparer la carte mémoire sur l'ordinateur

Étant donné que le Raspberry Pi ne peut pas encore démarrer tout seul, nous devons préparer préalablement la carte mémoire sur l'ordinateur. Pour ce faire, vous avez besoin d'un lecteur de cartes sur l'ordinateur. Ce dernier peut être intégré de manière fixe ou connecté via un port USB.

Il est recommandé d'utiliser une carte mémoire neuve car elle est déjà préformatée de manière optimale par le fabricant. Cependant, vous pouvez également utiliser une carte mémoire qui a déjà été utilisée dans un appareil photo numérique ou un autre appareil. Ces cartes mémoires doivent être à nouveau formatées avant d'être utilisées dans le Raspberry Pi. Pour ce faire, vous pouvez théoriquement utiliser les fonctionnalités de formatage de Windows. Il est préférable d'utiliser le logiciel « SDFormatter » de la SD Association. Il permet de formater les cartes mémoires pour une performance optimale. Vous pouvez télécharger gratuitement cet outil à l'adresse www.sdcard.org/downloads/formatter_4.

Si la carte mémoire contient des partitions d'une installation précédente de système d'exploitation pour le Raspberry Pi, sa capacité complète n'est pas affichée dans SDFormatter. Dans ce cas, utilisez l'option de formatage *FULL (Erase)* et activez l'option *Format Size Adjustment*. La partition de la carte mémoire est ensuite recréée.

La carte mémoire sera effacée

Nous vous recommandons d'utiliser une carte mémoire vide pour l'installation du système d'exploitation. S'il y a des données sur la carte mémoire, elles seront effacées irréversiblement au cours du processus de reformatage, pendant l'installation du système d'exploitation.

1.2.2 Installateur de logiciel NOOBS

Le « New Out Of Box Software » (NOOBS) est un installateur particulièrement simple pour le système d'exploitation du Raspberry Pi. L'utilisateur n'a plus besoin de traiter avec des outils image et des blocs de démarrage comme avant pour mettre en place une carte mémoire démarrable/amorçable. NOOBS propose plusieurs systèmes d'exploitation au choix, à partir desquels vous pouvez choisir le système d'exploitation souhaité lors du premier démarrage directement sur Raspberry Pi, qui est ensuite installé de manière démarrable/amorçable sur la carte mémoire. Téléchargez le fichier compressé d'env. 1,2 Go pour installer NOOBS, à partir du site officiel de téléchargement à l'adresse suivante www.raspberrypi.org/downloads, et décompressez-le sur l'ordinateur sur une carte mémoire d'au moins 4 Go.

Démarrez maintenant le Raspberry Pi avec cette carte mémoire. Pour ce faire, insérez-la dans l'emplacement dédié du Raspberry Pi puis branchez le clavier, la souris, le moniteur et le câble réseau. Connectez en dernier l'alimentation USB. Cela met en marche le Raspberry Pi. Il n'y a pas de bouton d'alimentation séparé.

Après quelques secondes, un menu s'affiche à partir duquel vous pouvez choisir le système d'exploitation souhaité. Nous utilisons le système d'exploitation Raspbian recommandé par la fondation Raspberry Pi.

Sélectionnez le [français] comme langue d'installation puis cochez le système d'exploitation Raspbian présélectionné. Après avoir confirmé le message de sécurité indiquant que la carte mémoire sera écrasée, l'installation se lance et est achevée en quelques minutes. Pendant l'installation, de courtes informations sur Raspbian s'affichent à l'écran.

1.2.3 Les voyants LED sur le Raspberry Pi

Cinq voyants LED indiquant le statut de Raspberry Pi se trouvent dans un coin de l'appareil. Les noms sont en partie différents sur les nouveaux et les anciens modèles de Raspberry Pi, mais les fonctions restent les mêmes.

Nouvelle carte de circuit imprimé (Rev. 2)	Carte de circuit imprimé (ancienne Rev. 1)	Couleur	Signification du voyant LED
ACT	OK	Vert	Accès sur la carte mémoire
PWR	PWR	Rouge	Connecté à l'alimentation électrique
FDX	FDX	Vert	LAN en mode duplex intégral
LNK	LNK	Vert	Accès sur le LAN
100	10M	Jaune.	LAN avec 100 Mo/s

Tableau 1.1: Voyants LED sur le Raspberry Pi

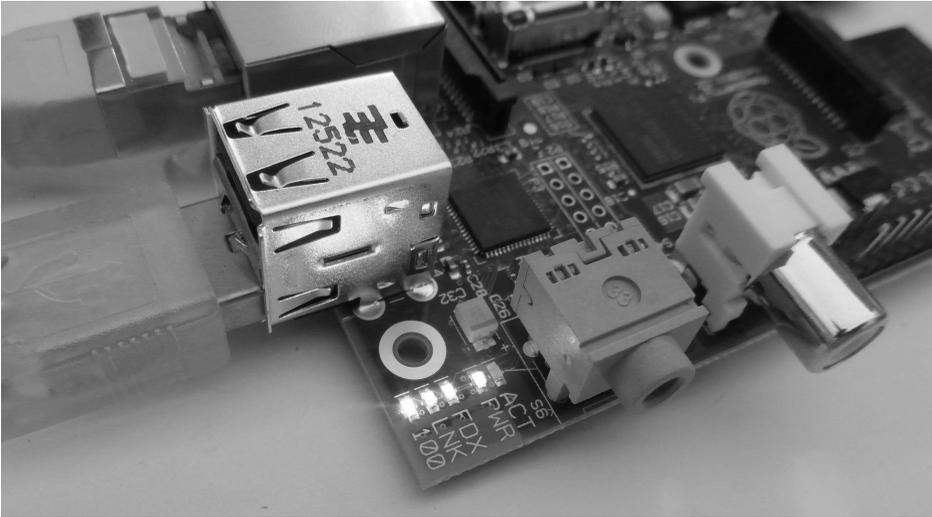


Fig. 1.2: Les voyants LED de statut sur le Raspberry Pi

1.2.4 Le premier démarrage sur le Raspberry Pi

À l'issue de l'installation, le Raspberry Pi redémarre et lance automatiquement l'outil de configuration `raspi-config`. Vous devez ici choisir uniquement dans *Enable Boot to Desktop* l'option *Desktop Log in as user 'pi'*. La langue [française] et le clavier [français] sont automatiquement sélectionnés ainsi que d'autres réglages importants. Après un redémarrage, le bureau graphique LXDE devient disponible.

1.3 Presque comme Windows - l'interface graphique LXDE

Le mot « Linux » fait peur à beaucoup de personnes, car elles craignent d'avoir à saisir des lignes de code comme il y a 30 ans sous DOS. Ce n'est pas le cas, loin de là ! En tant que système d'exploitation ouvert, Linux offre gratuitement aux développeurs un moyen de développer une interface graphique propre. Vous n'êtes donc pas lié(e) à une interface en tant qu'utilisateur du système d'exploitation toujours basé sur des lignes de commande dans le noyau.

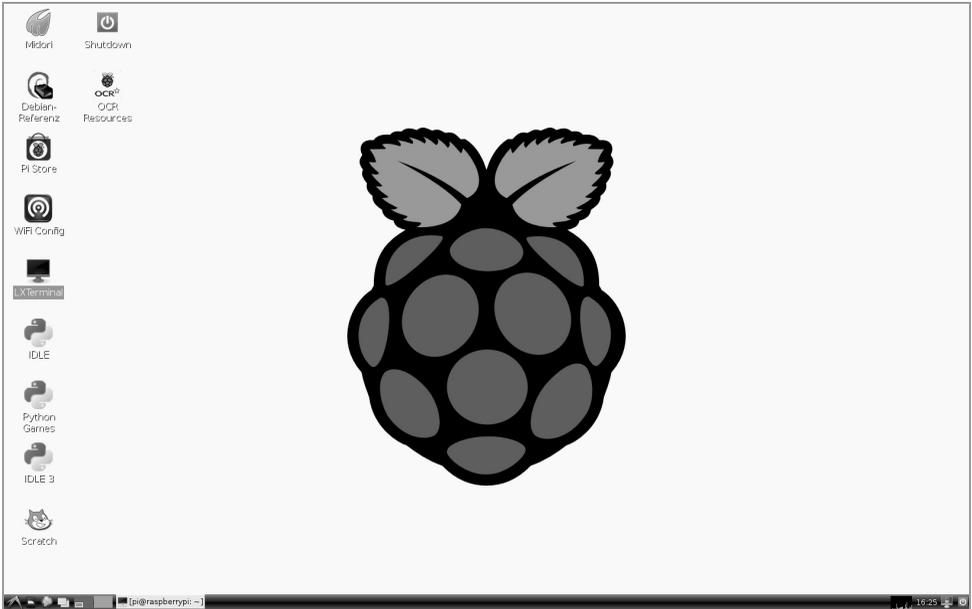


Fig. 1.3: Le Desktop LXDE (bureau LXDE) sur Raspberry Pi ressemble beaucoup à celui de Windows XP.

La version Raspbian de Linux pour le Raspberry Pi utilise l'interface LXDE (Lightweight X11 Desktop Environment), qui, d'une part, a besoin de très peu de ressource système et qui, d'autre part, ressemble fortement au menu de démarrage et au gestionnaire de fichiers de votre interface habituelle de Windows.

Application Linux

Même la connexion utilisateur caractéristique de Linux s'effectue en arrière-plan. Cependant, si vous en avez besoin : Le nom d'utilisateur est `pi` et le mot de passe `raspberrypi`.

Le symbole LXDE en bas à gauche ouvre le menu Démarrer, les symboles à côté ouvrent le gestionnaire de fichiers et le navigateur Internet. Le menu Démarrer est structuré sur plusieurs niveaux comme sous Windows. Les programmes fréquemment utilisés peuvent être sauvegardés avec un clic droit de souris sur le Desktop (bureau). Voici quelques programmes déjà préinstallés : le navigateur Internet Midori, l'environnement de développement Python et le Pi Store.

Mettre en arrêt le Raspberry Pi

On peut théoriquement débrancher simplement le Raspberry Pi pour qu'il s'éteigne. Il est cependant recommandé d'arrêter proprement le système comme un ordinateur classique. Pour ce faire, double-cliquez sur le symbole *Shutdown* qui se trouve sur le bureau.

1.3.1 Enregistrer vos fichiers sur le Raspberry Pi

La gestion des fichiers est un peu différente sur Linux par rapport à Windows mais cela n'est pas plus difficile. Raspbian offre un gestionnaire de fichiers qui ressemble beaucoup à l'explorateur de Windows. Cependant, il y a une différence importante par rapport à Windows : Linux ne sépare pas strictement en fonction des lecteurs. Tous les fichiers se trouvent dans un seul système de fichier unique.

Sous Linux, vous mettez en principe tous vos fichiers seulement sous le répertoire Home. Il s'appelle ici `/home/pi` pour le nom d'utilisateur `pi`. Linux utilise la barre oblique simple (`/`), pour séparer, et non la célèbre barre oblique inversée de Windows (`\`). Vous mettez également vos programmes Python dans ce répertoire. Le gestionnaire de fichiers, qui peut être ouvert via le menu Démarrer ou avec la combinaison de touche `Win` + `E` montre par défaut seulement ce répertoire Home. Certains programmes sont placés automatiquement dans des sous-répertoires.

Les personnes qui souhaitent tout voir, même les fichiers qui ne concernent pas l'utilisateur normal, changent le gestionnaire de fichiers en haut à gauche de la position *Emplacement* à *Arborescence du répertoire*. Choisissez ensuite dans le menu sous *Affichage* l'option *Affichage détaillé*, l'affichage apparaît comme l'on a préréglé sous Linux.

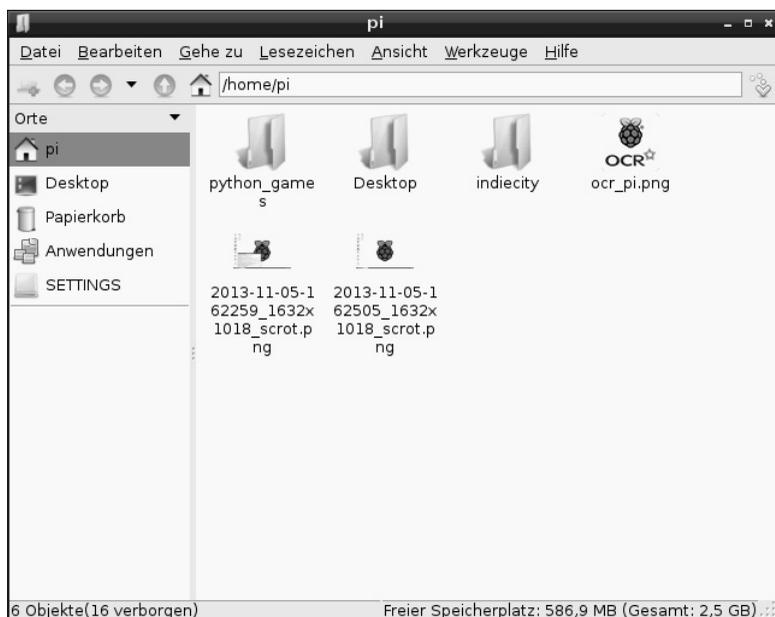


Fig. 1.4: Le gestionnaire de fichiers de Raspberry Pi peut ressembler à cela...

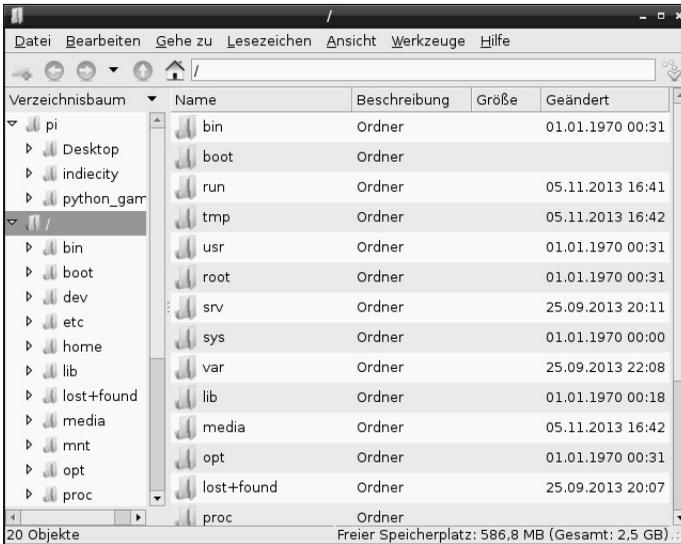


Fig. 1.5: ... ou à ceci.

Combien d'espace reste-t-il disponible sur la carte mémoire ?

Contrairement aux disques durs des ordinateurs qui sont en permanence pleins - grâce à sa carte mémoire, le Raspberry Pi peut fonctionner beaucoup plus vite. Il est d'autant plus important d'avoir toujours de l'espace libre et disponible sur la carte mémoire en un clin d'œil. La barre d'état du gestionnaire de programme en bas de la fenêtre indique à droite l'espace libre et disponible de la carte mémoire.

1.4 Le premier programme avec Python

Pour faire ses premiers pas dans la programmation, le langage de programmation Python est préinstallé sur le Raspberry Pi. Python se distingue par sa structure claire qui permet une introduction simple dans la programmation. C'est également une langue idéale pour « automatiser rapidement des tâches répétitives » que vous auriez pu autrement faire manuellement. Étant donné qu'aucune déclaration de variables, aucun type, aucune classe ou aucune règle compliquée ne doit être respectée, programmer devient vraiment amusant.

Python 2.7.3 ou 3.3.0 ?

Les deux versions de Python sont préinstallées sur Raspberry Pi. La nouvelle version de Python 3.x utilise malheureusement en partie une syntaxe différente par rapport à la version éprouvée 2.x, de sorte que les programmes créés avec une version ne marchent pas avec l'autre. Certaines bibliothèques importantes comme p. ex. la célèbre PyGame pour la programmation de jeux et de représentations graphiques générales, ne sont pas disponibles pour Python 3.x. Par conséquent, et aussi parce que la plupart des programmes disponibles sur Internet sont écrits pour Python 2.x, nous utilisons dans ce manuel la version éprouvée 2.7.3 de Python. Si une version de Python plus ancienne, avec un numéro de version 2.x est installée sur votre Raspberry Pi, nos exemples fonctionneront également.



Python 2.7.3 est lancé avec le symbole *IDLE* sur le bureau. En un clin d'œil, une fenêtre de saisie avec une invite de commande s'affiche ici à l'écran.

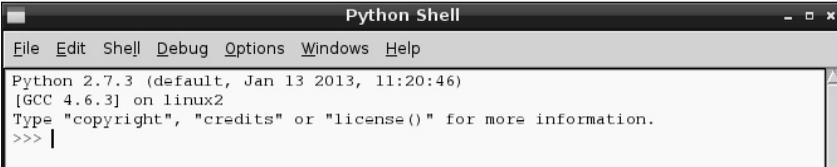


Fig. 1.6: La fenêtre de saisie du Python-Shell.

Dans cette fenêtre, vous ouvrez les programmes Python existants, vous en écrivez de nouveaux ou vous pouvez directement modifier les commandes Python de manière interactive, sans avoir à écrire un vrai programme. Entrez p. ex. dans l'invite de commande ce qui suit :

```
>>> 1+2
```

La réponse correcte s'affiche ensuite immédiatement :

```
3
```

De cette façon, Python peut être utilisé comme une calculatrice pratique mais cela n'a rien à voir avec la programmation. Le cours de programmation commence habituellement avec un programme *Bonjour le monde*, qui écrit sur l'écran la phrase « Bonjour le monde ». Cela est si facile avec Python qu'il n'est même pas nécessaire d'insérer son propre titre. Il vous suffit de taper dans l'écran du Python-Shell la phrase suivante :

```
>>> print "Bonjour le monde"
```

Ce premier « programme » écrit ensuite *Bonjour le monde* dans la ligne suivante à l'écran.

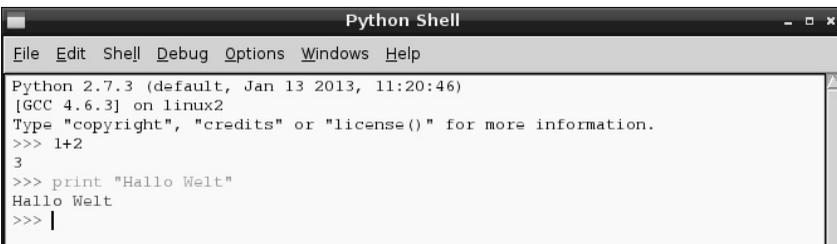


Fig. 1.7: « Bonjour le monde » en Python (La réponse du calcul est encore visible au-dessus).

Vous voyez également ici que le Python-Shell utilise automatiquement différentes couleurs de texte à des fins de clarification. Les commandes Python sont en orange, les chaînes de caractères en vert et les résultats en bleu. Vous découvrez plus tard encore d'autres couleurs.

Cartes flash Python

Python est le langage de programmation idéal pour commencer à apprendre à programmer. Seule la syntaxe et les règles de mises en page sont un peu étranges. Pour vous aider à programmer, les éléments importants de la syntaxe du langage Python sont décrits brièvement sous la forme « d'antisèche ». Elle repose sur les cartes flash de David Whale. Vous trouverez tout ce dont vous avez besoin à l'adresse bit.ly/pythonflashcards. Ces cartes flash n'expliquent pas le contexte technique mais seulement décrivent la syntaxe avec des exemples brefs et comment faire quelque chose.

1.4.1 Deviner les chiffres avec Python

Plutôt que de nous arrêter sur les principes de la programmation, les algorithmes et les types de données, nous préférons écrire le premier petit jeu dans Python, une devinette simple, dans laquelle un chiffre choisi aléatoirement par l'ordinateur doit être deviné par le joueur avec le moins d'étapes possibles.

1. Choisissez dans le menu le Python-Shell *File/New Window*. Une nouvelle fenêtre s'ouvre ici, dans laquelle vous entrez le code suivant :

```
import random
chiffre = random.randrange(1000); réponse = 0; i = 0
while réponse != chiffre:
    réponse = input("Ta réponse :")
    if chiffre < réponse:
        print "Le chiffre cherché est plus petit que",réponse

    if chiffre > réponse:
        print "Le chiffre cherché est plus grand que ",réponse

    i += 1
print "Vous avez trouvé le chiffre à la ",i,"e réponse"
```

2. Enregistrez le fichier avec *File/Save As* en tant que `spiel1.py`. Ou téléchargez le fichier fini du programme à partir de www.buch.cd et ouvrez-le dans Python-Shell avec *File/Open*. Le code couleur dans le code source apparaît automatiquement et aide à trouver les erreurs de saisie.
3. Avant de lancer le jeu, vous devez tenir compte d'une particularité de la langue [française], à savoir les [accents]. Python fonctionne sur différentes plateformes informatiques qui codent les [accents] différemment. Pour qu'ils soient correctement affichés, choisissez dans le menu *Options/Configure IDLE* et activez sur l'onglet *General* l'option *Locale-defined* dans le champ *Default Source Encoding*.

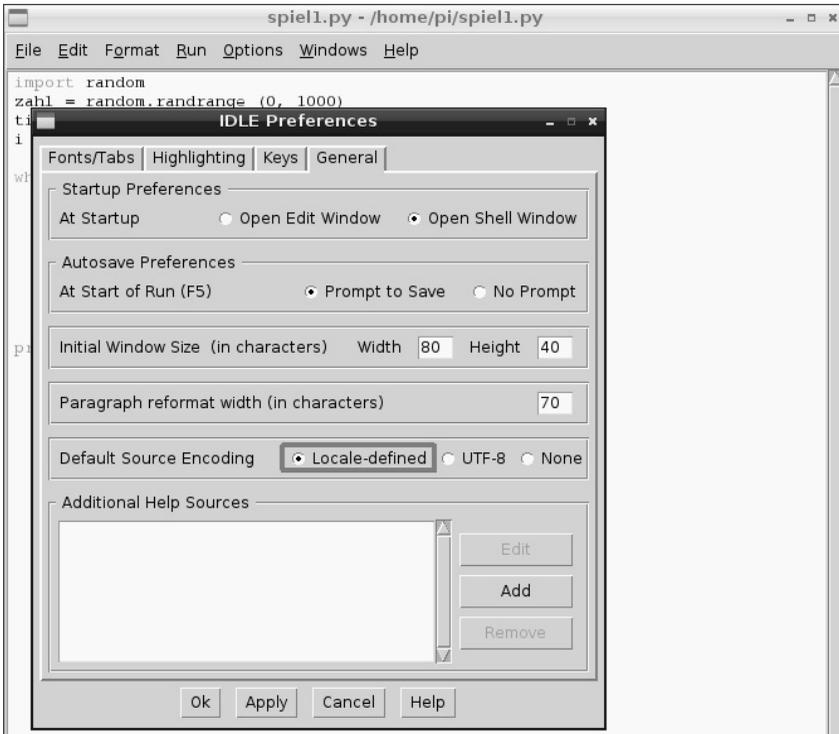
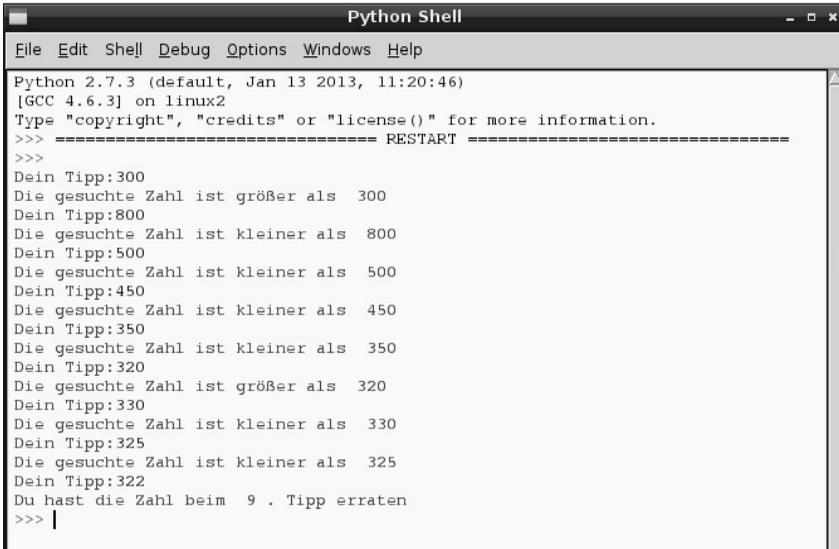


Fig. 1.8: Le réglage correct pour afficher les [accents] dans Python.

4. Lancez maintenant le jeu avec la touche `[F5]` ou le point de menu *Run/Run Module*.
5. Par souci de simplicité, le jeu renonce à l'interface graphique, au texte explicatif ou aux questions de plausibilité de la réponse. L'ordinateur génère en arrière-plan un chiffre aléatoire compris entre 0 et 1000. Il suffit d'entrer votre réponse et vous saurez si le chiffre cherché est plus grand ou plus petit. En plusieurs réponses, vous vous approchez du bon chiffre.



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Dein Tipp:300
Die gesuchte Zahl ist größer als 300
Dein Tipp:800
Die gesuchte Zahl ist kleiner als 800
Dein Tipp:500
Die gesuchte Zahl ist kleiner als 500
Dein Tipp:450
Die gesuchte Zahl ist kleiner als 450
Dein Tipp:350
Die gesuchte Zahl ist kleiner als 350
Dein Tipp:320
Die gesuchte Zahl ist größer als 320
Dein Tipp:330
Die gesuchte Zahl ist kleiner als 330
Dein Tipp:325
Die gesuchte Zahl ist kleiner als 325
Dein Tipp:322
Du hast die Zahl beim 9 . Tipp erraten
>>> |
```

Fig. 1.9: Deviner le chiffre dans Python.

1.4.2 Voilà comment cela fonctionne

Comme le jeu fonctionne, vous pouvez l'essayer facilement. Vous vous posez naturellement quelques questions maintenant : Qu'est-ce qu'il se passe en arrière-plan ? Qu'est-ce que ces lignes de programme signifient ?

`import random` Pour générer le chiffre aléatoire, un module de Python externe nommé `random` est importé. Il contient différentes fonctions nécessaires pour générer le chiffre aléatoire.

`chiffre = random.randrange(1000)` La fonction `randrange` du module `random` génère un chiffre aléatoire dans la gamme délimitée par les paramètres, ici entre 0 et 999. Le paramètre de la fonction `random.randrange()` spécifie le nombre de chiffres aléatoires possibles, commençant par 0 donc toujours le premier chiffre qui n'est pas atteint. Il en est de même pour les boucles et les fonctions semblables dans Python.

Ce chiffre aléatoire est enregistré dans les variables `chiffre`. Les variables sont les emplacements mémoires dans Python, qui portent un nom donné et qui peuvent enregistrer des chiffres, des chaînes de caractères, des listes ou d'autres types de données. Contrairement à d'autres langages de programmation, vous ne devez pas les déclarer préalablement.

Comment les chiffres aléatoires sont-ils générés ?

On pense communément que rien ne peut se produire au hasard dans un programme. Alors, comment est-ce qu'un programme peut générer un chiffre aléatoire ? Si l'on divise un grand nombre premier par n'importe quel chiffre, on obtient des chiffres à x décimales qui peuvent être devinés difficilement. Ils changent sans aucune régularité lorsque l'on augmente régulièrement le diviseur. Bien que ce résultat soit apparemment dû au hasard, il peut être reproduit à tout moment par un programme identique ou en appelant plusieurs fois le même programme. Si l'on prend maintenant un nombre obtenu à partir de certains de ces chiffres et que l'on le divise à nouveau par un chiffre qui découle de la seconde du temps actuel ou du contenu d'un emplacement mémoire donné de l'ordinateur, on obtient un résultat qui n'est pas reproductible et qui est alors appelé un chiffre aléatoire.

`réponse = 0` La variable `réponse` contient le nombre que l'utilisateur a donné. Au début, elle est 0.

`i = 0` La variable `i` s'est établie chez les développeurs comme le compteur pour les cycles de boucles du programme. Elle est utilisée ici pour compter le nombre de réponses que l'utilisateur a donné pour deviner le chiffre secret. Cette variable est aussi égale à 0 au début.

`while réponse != chiffre:` Le mot `while` (en anglais signifiant « tant que ») lance une boucle de programme qui est répétée dans ce cas tant que la `réponse`, le chiffre que l'utilisateur donne n'est pas égal au chiffre secret `chiffre`. Python utilise la combinaison de caractères `!=` pour l'inégalité. La boucle de programme s'exécute en fait après les deux points (:).

`réponse = input("Ta réponse :")` La fonction `input` écrit le texte `Ta réponse :` puis attend ensuite une commande qui est enregistrée dans la variable `réponse`.

Les indentations sont importantes dans Python

Dans la plupart des langages de programmation, les boucles de programme ou les décisions utilisent l'indentation [ajout de tabulation ou d'espaces dans un fichier texte] pour faciliter la lecture du code du programme. Dans Python, cette indentation sert non seulement à des fins de clarification mais elle est également absolument nécessaire pour la logique du programme. On n'a pas besoin ici de ponctuation spéciale pour quitter les boucles ou les décisions.

`if chiffre < réponse:` Si le chiffre secret `chiffre` est plus petit que le chiffre `réponse` donné par l'utilisateur, alors...

```
print "Le chiffre cherché est plus petit que", réponse
```

... ce texte est délivré. À la fin, la variable `réponse` est utilisée pour que le chiffre donné apparaisse dans le texte. Si cela ne répond pas à cette condition, la ligne indentée est simplement ignorée.

`if réponse < chiffre:` Si le chiffre secret `chiffre` est plus grand que le chiffre `réponse` donné par l'utilisateur, alors...

```
print "Le chiffre cherché est plus grand que", réponse
```

... un autre texte est délivré.

$i += 1$ Dans chaque cas - donc plus indenté - le compteur i , qui compte le nombre de tentatives, est augmenté de 1. Cette ligne avec l'opérateur $+=$ veut dire la même chose que $i = i + 1$.

```
print "Vous avez trouvé le chiffre à la ",i,"e réponse"
```

Cette ligne est plus indentée ce qui signifie que la boucle `while` est également quittée. Si la condition n'est plus satisfaite, que le chiffre `réponse` donné par l'utilisation n'est plus inégal (mais égal) au chiffre secret `chiffre`, ce texte est délivré, qui se compose des deux parties de la phrase et des variables i et qui indique le nombre de tentatives nécessaires pour que l'utilisateur trouve la bonne réponse. Les programmes Python n'ont besoin d'aucune instruction en particulier pour se terminer. Ils se terminent tout simplement après la dernière ligne de commande ou après une boucle qui n'est plus exécutée et après laquelle il n'y a pas d'autre instruction.

2 Le premier voyant LED s'allume sur le Raspberry Pi

La barrette à 26 broches dans le coin du Raspberry Pi permet de connecter directement des périphériques pour p. ex. saisir des entrées via des boutons ou allumer les voyants LED contrôlés par un programme. Cette barrette est également appelée GPIO. L'acronyme anglais de « General Purpose Input Output » signifie simplement en [français] « Entrée et sortie d'usage général ».

17 de ces 26 broches peuvent être programmées comme des entrées ou des sorties comme vous le souhaitez, et elles peuvent être utilisées pour de nombreuses extensions matérielles. Les autres broches sont dédiées à l'alimentation électrique et à d'autres fins.

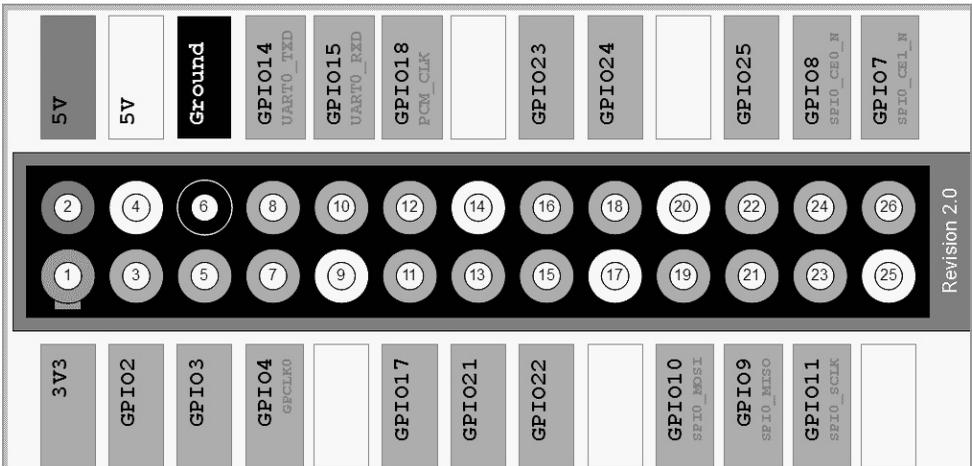


Fig. 2.1: Attribution des interfaces GPIO La ligne grise en haut et à gauche indique le bord de la carte de circuit imprimé. La broche GPIO 2 se trouve complètement à l'extérieur, dans le coin du Raspberry Pi.

Attention !

En aucun cas, vous ne devez connecter les broches GPI entre elles et attendre la suite des événements. Suivez impérativement les instructions suivantes :

Certaines broches GPIO sont directement connectées aux bornes du processeur et un court-circuit peut endommager irréversiblement le Raspberry Pi. Si vous connectez deux broches ensemble via un interrupteur ou une LED, il faut toujours intercaler une pré-résistance.

Utilisez toujours pour les signaux logistiques la broche 1 qui délivre une tension de + 3,3 V et qui peut supporter jusqu'à 50 mA. La broche 6 est le fil de terre pour les signaux logistiques. Les autres broches, 14, 17, 20, 25 marquée par *Ground* ou *3V3* sont réservées aux extensions ultérieures. Vous pouvez actuellement les utiliser comme indiqué. Cependant, vous ne devez pas le faire pour pouvoir les utiliser pour vos projets aussi avec les versions à venir du Raspberry Pi.

Chaque broche GPIO peut être définie comme une sortie (p. ex. pour les LED) ou comme une entrée (p. ex. pour les boutons). Les sorties GPIO délivrent en état logique *1* une tension de + 3,3 V, en état logique *0* 0 volt. Les sorties GPIO délivrent sous une tension allant jusqu'à + 1,7 V le signal logique *0*, sous une tension comprise entre + 1,7 V et + 3,3 V le signal logique *1*.

La broche 2 délivre une tension de + 5 V pour alimenter un périphérique externe. Le courant délivré peut être aussi élevé que celui délivré par le bloc d'alimentation USB du Raspberry Pi. Cette broche ne doit cependant pas être connectée avec une entrée GPIO.

2.1 Composants dans le kit

Le kit d'apprentissage contient différents composants électroniques, qui vous permettent de réaliser les expériences décrites (et naturellement les vôtres). A ce stade, les composants sont décrits uniquement brièvement. L'expérience pratique qui est nécessaire pour les utiliser est ensuite apportée par les expériences réelles.

- 2x Cartes de circuit imprimé
- 1x LED rouge
- 1x LED jaune
- 1x LED verte
- 1x LED bleue
- 4x Boutons
- 4x Résistances de 10 k Ω (Marron-Noir-Orange)
- 4x Résistances de 1 k Ω (Marron-Noir-Rouge)

- 4x Résistances de 220 Ω (Rouge-Rouge-Marron)
- 12x Câbles de connexion (Circuit imprimé - Raspberry Pi)
- fil de connexion d'env. 1 m

2.1.1 Cartes de circuits imprimés

Deux cartes de circuit imprimé sont incluses dans le kit pour construire rapidement des circuits électroniques. Les composants électroniques peuvent être directement branchés sur une trame à perforation avec des distances standard, sans avoir à les souder. Dans ces cartes, les rangées longitudinales extérieures avec les contacts (X et Y) sont toutes interconnectées.

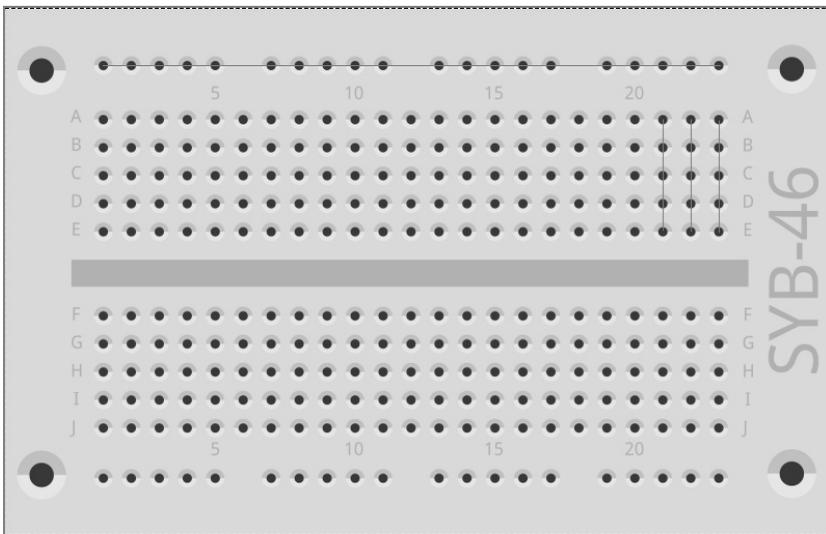


Fig. 2.2: La carte de circuit imprimé du kit avec quelques composants montrés pour l'exemple.

Ces rangées de contact sont souvent utilisées comme les pôles plus et moins pour l'alimentation électrique des circuits. Dans les autres rangées de contact, 5 contacts sont interconnectés (A à E et F à J) là où il y a un trou au centre de la carte. De grands composants peuvent être insérés au centre et câblés vers l'extérieur.

2.1.2 Câble de connexion

Les câbles de connexion de couleur ont tous sur l'un des côtés un petit connecteur de fils qui peut être inséré sur la carte de circuit imprimé. De l'autre côté, il y a une prise qui s'adapte sur une broche GPIO du Raspberry Pi.

En outre, le fil de connexion est inclus dans le kit d'apprentissage. Il permet de créer des ponts de jonction courts pour connecter les rangées de contact sur la carte de circuit imprimé. Coupez la longueur appropriée

de fil avec un petit coupe-fil comme décrit dans chaque expérience. Pour pouvoir mieux insérer le fil dans la carte de circuit imprimé, il est recommandé de couper le fil de manière légèrement oblique de sorte à créer une sorte de pointe. Dénudez la gaine isolante aux deux extrémités sur une longueur d'environ un demi-centimètre.

2.1.3 Résistances et leur code couleur

Les résistances sont utilisées dans l'électronique numérique essentiellement pour limiter le courant arrivant au port d'un microcontrôleur mais également comme pré-résistance pour les LED. L'unité de mesure de la résistance est l'ohm (Ω). 1000 ohms correspondent à un kilohm, qui est abrégé par $k\Omega$.

La valeur de la résistance est donnée sur la résistance par des anneaux de couleur. La plupart des résistances ont quatre anneaux de couleur. Les deux premiers anneaux de couleur indiquent le chiffre, le troisième un multiplicateur et le quatrième la tolérance. Cet anneau de tolérance est la plupart du temps de couleur or - ou argent. Le premier anneau ne peut pas être de ces couleurs de sorte que le sens de lecture est clair. La valeur de la tolérance ne joue en soi aucun rôle dans l'électronique numérique.

Couleur	Valeur de la résistance en ohm (Ω)			
	1. Anneau (1er chiffre ; dizaine)	2. Anneau (2e chiffre)	3. Anneau (Multiplicateur)	4. Anneau (Tolérance)
Argent			$10^{-2} = 0,01$	$\pm 10 \%$
Or			$10^{-1} = 0,1$	$\pm 5 \%$
Noir.		0	$10^0 = 1$	
Marron	1	1	$10^1 = 10$	$\pm 1 \%$
Rouge	2	2	$10^2 = 100$	$\pm 2 \%$
Orange.	3	3	$10^3 = 1\ 000$	
Jaune.	4	4	$10^4 = 10\ 000$	
Vert	5	5	$10^5 = 100\ 000$	$\pm 0,5 \%$
Bleu	6	6	$10^6 = 1\ 000\ 000$	$\pm 0,25 \%$
Violet	7	7	$10^7 = 10\ 000\ 000$	$\pm 0,1 \%$
Gris	8	8	$10^8 = 100\ 000\ 000$	$\pm 0,05 \%$
Blanc	9	9	$10^9 = 1\ 000\ 000\ 000$	

Tableau 2.1: Le tableau montre la signification des anneaux de couleur sur les résistances.

Le kit d'apprentissage contient des résistances de trois valeurs différentes :

Valeur	1. Anneau (1er chiffre ; dizaine)	2. Anneau (2e chiffre)	3. Anneau (Multipliat eur)	4. Anneau (Toléranc e)	Usage
220 Ω	Rouge	Rouge	Marron	Or	Pré-résistance pour LED
1 k Ω	Marron	Noir.	Rouge	Or	Résistance de protection pour les entrées GPIO
10 k Ω	Marron	Noir.	Orange.	Or	Résistance Pull-Down pour les entrées GPIO

Tableau 2.2: Codes couleur des résistances incluses dans le kit d'apprentissage

Faites particulièrement attention aux couleurs des résistances de 1 k Ω et de 10 k Ω . Il est facile de les intervertir.

2.2 Brancher les LED

Les voyants LED (LED = Light Emitting Diode, soit en français = diode électroluminescente) pour les signaux lumineux et les effets lumineux peuvent être branchés sur les ports GPIO. Pour ce faire, une pré-résistance de 220 Ω (Rouge-Rouge-Marron) doit être montée entre la broche GPIO utilisée et l'anode de la LED, pour limiter le courant passant et éviter de griller la LED. En outre, la pré-résistance protège également la sortie GPIO du Raspberry Pi. En effet, la LED n'offre quasiment aucune résistance dans le sens du courant et le port GPIO peut donc être surchargé rapidement s'il est connecté à la masse. On branche la cathode de la LED avec le fil de masse sur la broche 6.

Dans quel sens faut-il brancher la LED ?

Les deux conducteurs de connexion d'une LED ont des longueurs différentes. Le conducteur le plus long des deux est le pôle plus, c.-à-d. l'anode, le plus court est la cathode. Astuce pour retenir facilement : Le signe plus a un trait en plus par rapport au signe moins, ce qui fait que le fil est un peu plus long. En outre, la plupart des LED sont aplaties du côté moins, ressemblant alors au signe moins. Astuce pour retenir facilement : Cathode = bord court

En premier lieu, branchez comme sur la figure une LED via une pré-résistance de 220 Ω (Rouge-Rouge-Marron) sur la prise + 3,3 V (broche 1) et connectez le pôle moins de la LED à la ligne de terre (broche 6).

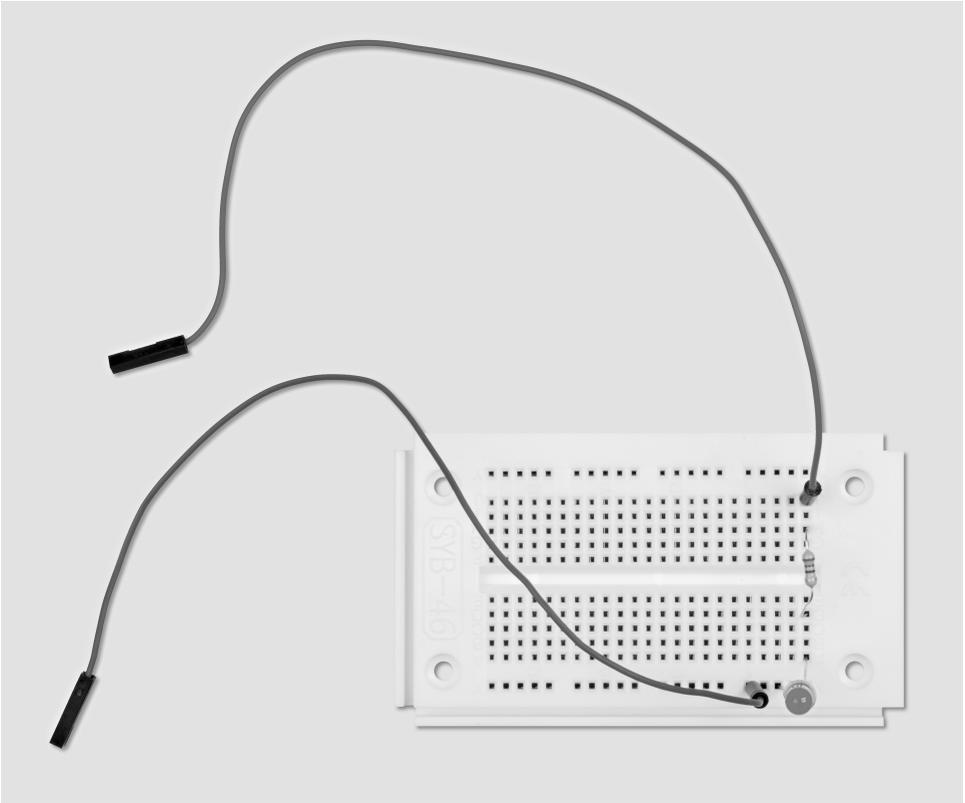


Fig. 2.3: Construction de la carte pour connecter une LED.

Composants nécessaires:
1x Carte de circuit imprimé
1x LED rouge
1x résistance de 220 Ω
2x Câbles de connexion

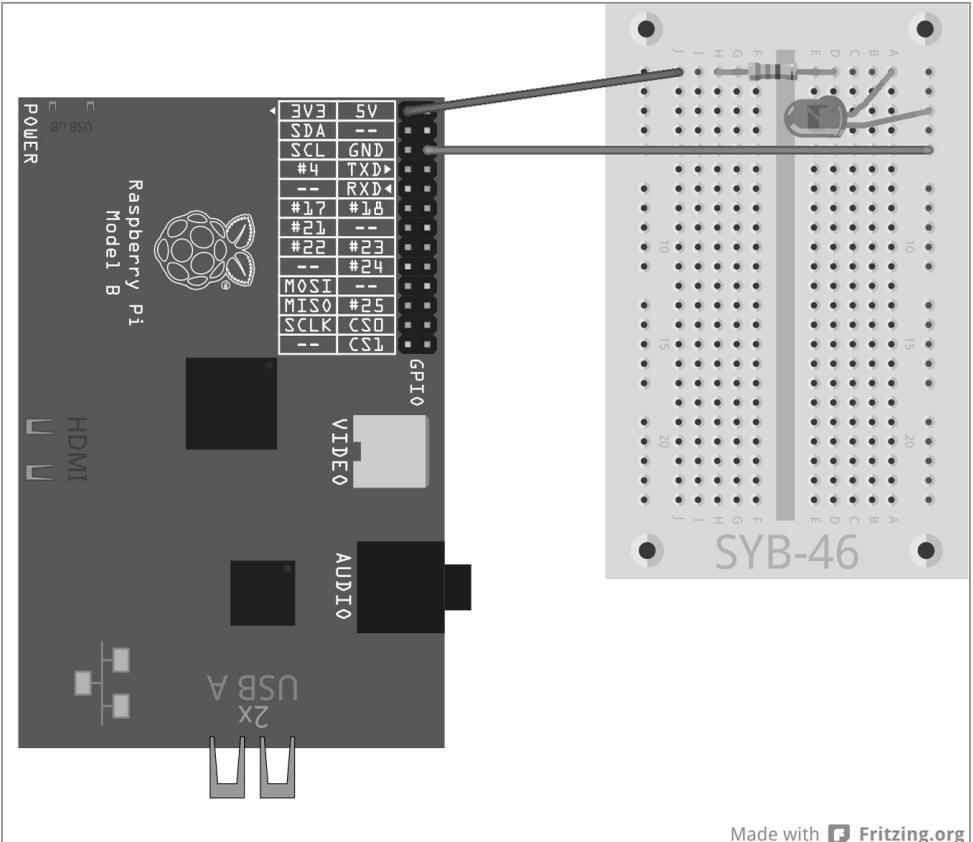


Fig. 2.4: Le premier voyant LED sur le Raspberry Pi

Dans cette première expérience, le Raspberry Pi est uniquement utilisé comme alimentation électrique de la LED. La LED s'allume toujours sans avoir besoin de logiciel.

Dans l'expérience suivante, vous installerez un bouton dans la ligne d'alimentation de la LED. La LED s'allume maintenant uniquement si vous appuyez sur le bouton. Aucun logiciel n'est également requis ici.

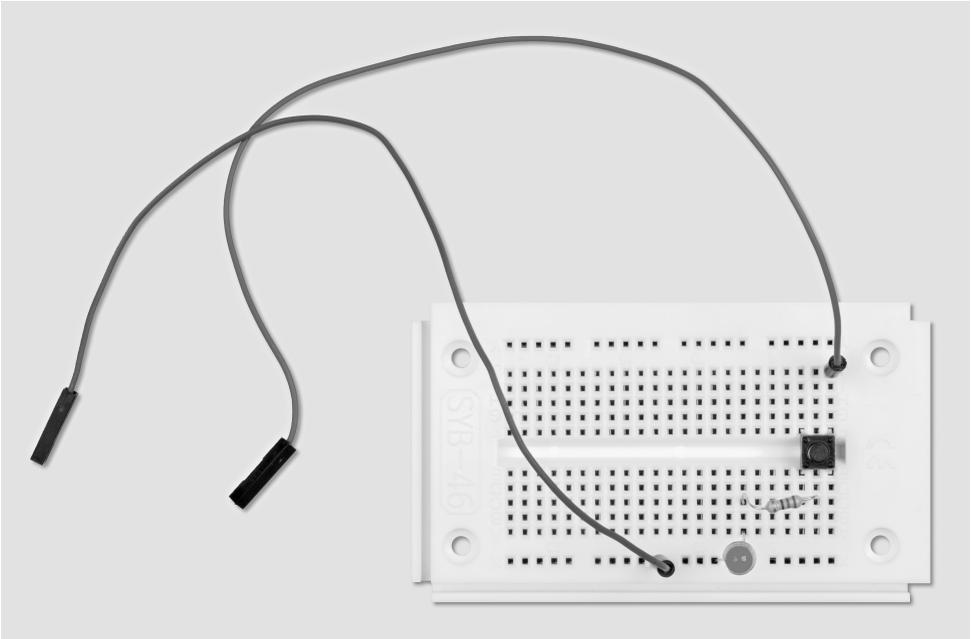


Fig. 2.5: Construction de la carte pour une LED qui est allumée avec un bouton

Composants nécessaires :
1x Carte de circuit imprimé
1x LED rouge
1x résistance de 220 Ω
1x Bouton
2x Câbles de connexion

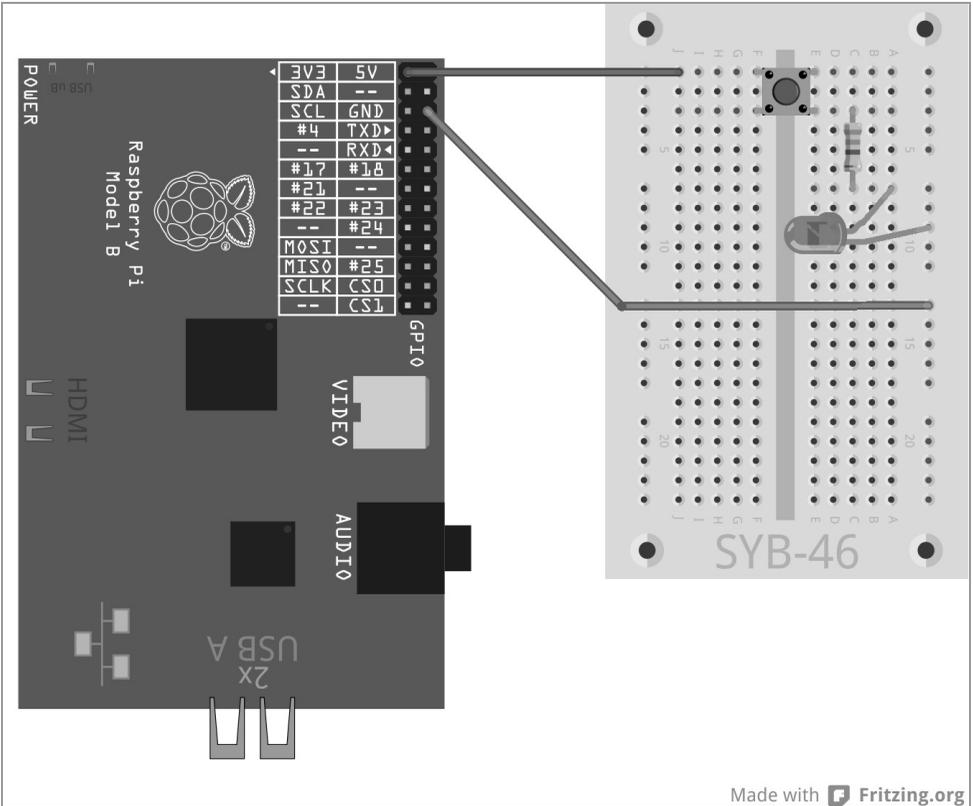


Fig. 2.6: LED avec bouton sur le Raspberry Pi.

2.3 GPIO avec Python

Pour pouvoir utiliser les ports GPIO avec un programme Python, vous devez installer la bibliothèque Python-GPIO. Si vous n'êtes pas certain(e) si tous les modules nécessaires sont installés, installez une fois les versions actuelles avec les commandes suivantes :

```
sudo apt-get update
sudo apt-get install python-dev
sudo apt-get install python-rpi.gpio
```

Comme d'habitude pour tous les périphériques sous Linux, les ports GPIO sont inclus comme des fichiers dans la structure du répertoire. Vous avez besoin des privilèges Root pour accéder à ces fichiers. Lancez également le Python-Shell avec les privilèges Root via un LXTerminal : `sudo idle`

2.4 Allumer et éteindre une LED

Branchez, comme indiqué sur la figure, une LED via une pré-résistance de 220 Ω (Rouge-Rouge-Marron) sur le port GPIO 25 (broche 22) et non plus directement sur la prise de + 3,3 V. Connectez le pôle moins de la LED au fil de terre du Raspberry Pi (broche 6) via le rail de masse de la carte de circuit imprimé.

Composants nécessaires :
1x Carte de circuit imprimé
1x LED rouge
1x résistance de 220 Ω
2x Câbles de connexion

Le programme suivant `led.py` allume la LED pendant 5 secondes puis l'éteint :

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.OUT)
GPIO.output(25, 1)
time.sleep(5)
GPIO.output(25, 0)
GPIO.cleanup()
```

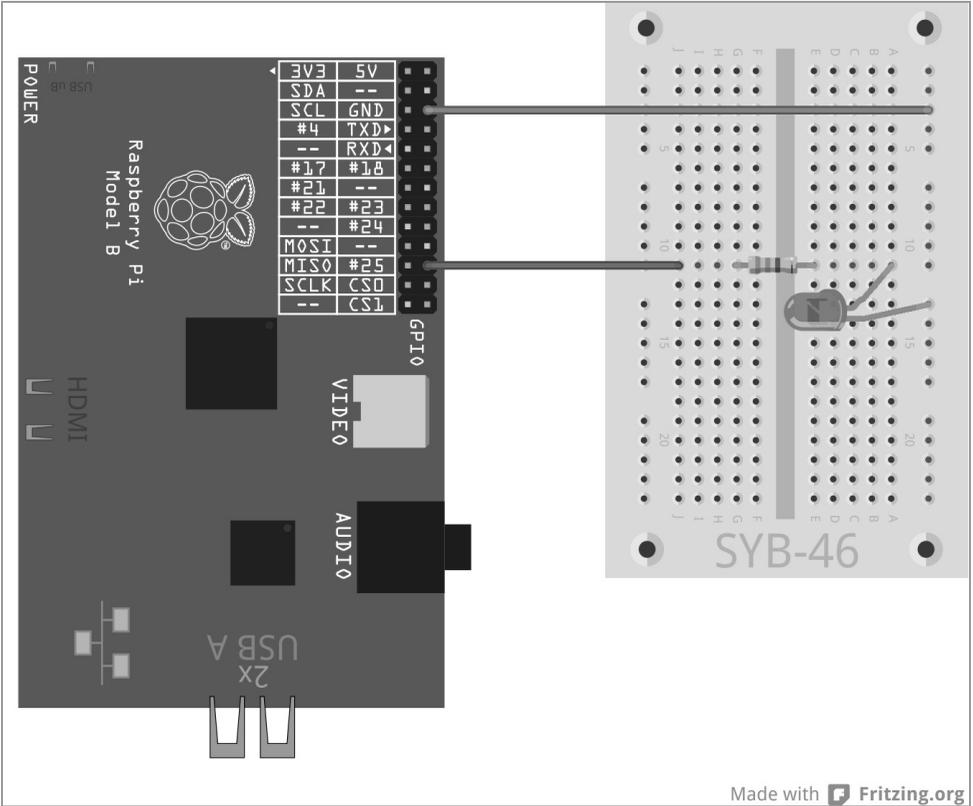


Fig. 2.7: Une LED sur le port GPIO 25

2.4.1 Voilà comment cela fonctionne

L'exemple montre les fonctions de base les plus importantes de la bibliothèque `RPi.GPIO`.

`import RPi.GPIO as GPIO` La bibliothèque `RPi.GPIO` doit être importée dans chaque programme Python où elle doit être utilisée. Cette manière d'écrire permet de consulter toutes les fonctions de la bibliothèque via le préfixe `GPIO`.

`import time` La fonction `time` couramment utilisée de la bibliothèque Python n'a rien à voir avec la programmation de GPIO. Elle comprend des fonctions pour le calcul de l'heure et de la date, y compris une fonction `time.sleep`, avec laquelle vous pouvez définir facilement des temps d'attente dans un programme.

`GPIO.setmode(GPIO.BCM)` Au début de chaque programme, il faut définir comment sont affectés les ports GPIO. On utilise normalement la numérotation standard BCM.

Numérotation des ports GPIO

La bibliothèque `Rpi.GPIO` supporte différentes méthodes pour affecter les ports. Dans le mode `BCM`, les numéros de port GPIO connus sont utilisés, qui peuvent également être utilisés à partir des lignes de commande ou dans les scripts de Shell. Dans le mode alternatif `BOARD`, l'affectation des numéros de broches de 1 à 26 correspondent à la carte Raspberry Pi.

`GPIO.setup(25, GPIO.OUT)` La fonction `GPIO.setup` initialise un port GPIO comme une sortie ou une entrée. Le premier paramètre affecte le port en fonction du mode prédéfini `BCM` ou `BOARD` avec son numéro GPIO ou son numéro de broche. Le deuxième paramètre peut être soit `GPIO.OUT` pour une sortie soit `GPIO.IN` pour une entrée.

`GPIO.output(25, 1)` Sur un port qui vient être initialisé, un 1 est donné. La LED qui y est branchée, s'allume. A la place de 1, les valeurs prédéfinies `True` ou `GPIO.HIGH` peuvent également être données.

`time.sleep(5)` Cette fonction entraîne un temps d'attente de 5 secondes à partir de la bibliothèque `time` importée au début, avant que le programme ne se poursuive.

`GPIO.output(25, 0)` Pour éteindre la LED, on transmet la valeur 0, `False` ou `GPIO.LOW` sur le port GPIO.

`GPIO.cleanup()` A l'issue d'un programme, tous les ports GPIO doivent être réinitialisés. Cette ligne traite en une fois tous les ports GPIO initialisés par le programme. Les ports qui sont initialisés par d'autres programmes restent inchangés. À la fin, les autres programmes potentiellement exécutés en parallèle ne sont pas affectés.

Intercepter les avertissements GPIO

Si un port GPIO doit être configuré sans avoir été réinitialisé correctement, mais qu'il a été ouvert par un autre programme ou par un programme corrompu, il est possible que des messages d'alerte soit émis sans interrompre cependant le déroulement du programme. Au cours du développement du programme, ces avertissements peuvent être très utiles pour détecter les erreurs. Dans un programme fini, ils peuvent déconcerter un utilisateur inexpérimenté. Par conséquent, la bibliothèque GPIO offre avec `GPIO.setwarnings(False)` la possibilité de supprimer ces avertissements.

2.5 Lancer Python avec support GPIO sans terminal

Les personnes qui bricolent beaucoup avec Python et GPIO ne souhaitent pas ouvrir à chaque fois un LXTerminal pour lancer l'IDLE. Il existe un moyen plus facile. Placez ce symbole sur le bureau qui peut ouvrir l'IDE de Python avec les privilèges de super-utilisateur :

- Créez une copie du symbole de bureau *IDLE* préinstallé. Pour ce faire, procédez comme suit :



- Faites un clic droit de souris sur le symbole *IDLE* du bureau et sélectionnez *Copier* dans le menu contextuel.

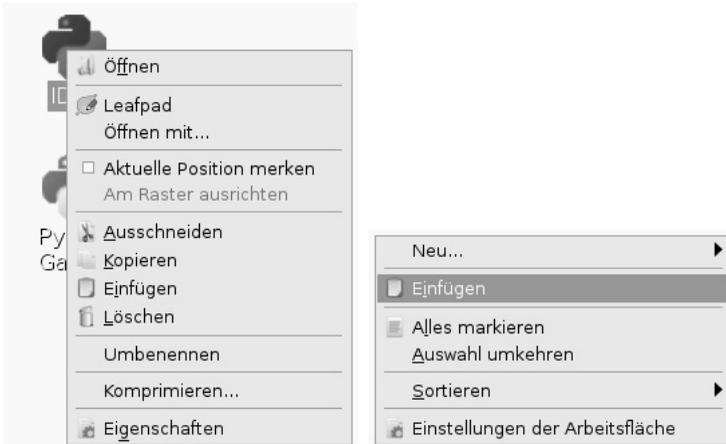


Fig. 2.8: Copier le symbole *IDLE* du bureau.

Ensuite faites un clic droit de souris sur le bureau et sélectionnez *Coller* dans le menu contextuel. Comme il existe déjà un raccourci bureau avec le même nom, un message indiquant que vous essayez de faire une copie s'affiche à l'écran.

Modifiez ici le nom de la copie de *idle.desktop* à *idle_gpio.desktop*. Rien ne change au premier abord au niveau du symbole sur le bureau. Le nom affiché reste *IDLE*.



Fig. 2.9: Message affiché lors de la duplication d'un raccourci bureau

Faites maintenant un clic droit de souris sur la copie du symbole de bureau et sélectionnez *Leafpad* dans le menu contextuel. Les raccourcis bureau sont des fichiers texte dans Linux, qui peuvent être modifiés à l'aide d'un éditeur de texte.



Fig. 2.10: Le raccourci bureau dans l'éditeur de texte Leafpad

Effectuez ici les deux modifications montrées dans la figure :

- Modifiez le champ `Name` en `IDLE GPIO`. Il s'agit du nom qui sera affiché à l'écran.
- Insérez dans le champ `Exec` avant la commande le mot `sudo`.

Fermez l'éditeur et enregistrez le fichier. En double-cliquant sur le nouveau symbole de bureau, lancez l'IDE de Python *IDLE* avec les privilèges de super-utilisateur. Vous pouvez maintenant utiliser les fonctions GPIO sans ouvrir Python via un LXTerminal.

3 Feu de signalisation

Allumer et éteindre une seule LED peut être amusant au début mais cela ne nécessite pas vraiment un ordinateur pour le faire. Un feu de signalisation avec son cycle de lumière typique passant du vert au jaune, du jaune au rouge puis une combinaison de rouge-jaune pour revenir au vert, est facile à réaliser avec trois LED et permet de montrer des techniques supplémentaires de programmation dans Python.

Montez le circuit montré dans la figure sur la carte de circuit imprimé. Pour contrôler les LED, trois ports GPIO et un fil de terre commun sont utilisés. Les numéros de ports GPIO en mode BCM sont montrés sur le Raspberry Pi dans la figure.

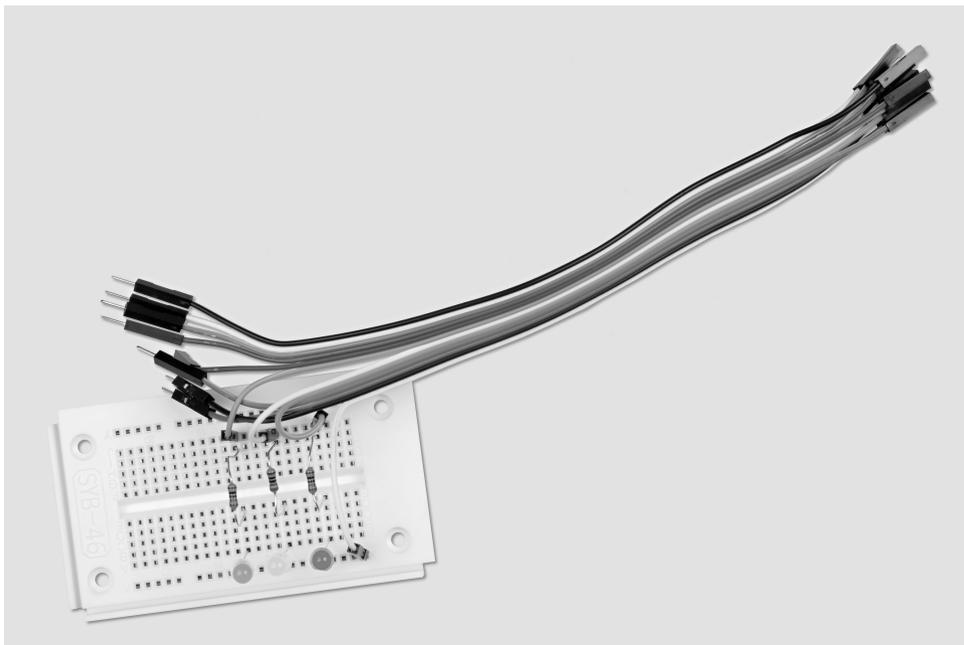


Fig. 3.1: Construction de la carte pour le feu de signalisation

Composants nécessaires :

1x Carte de circuit imprimé

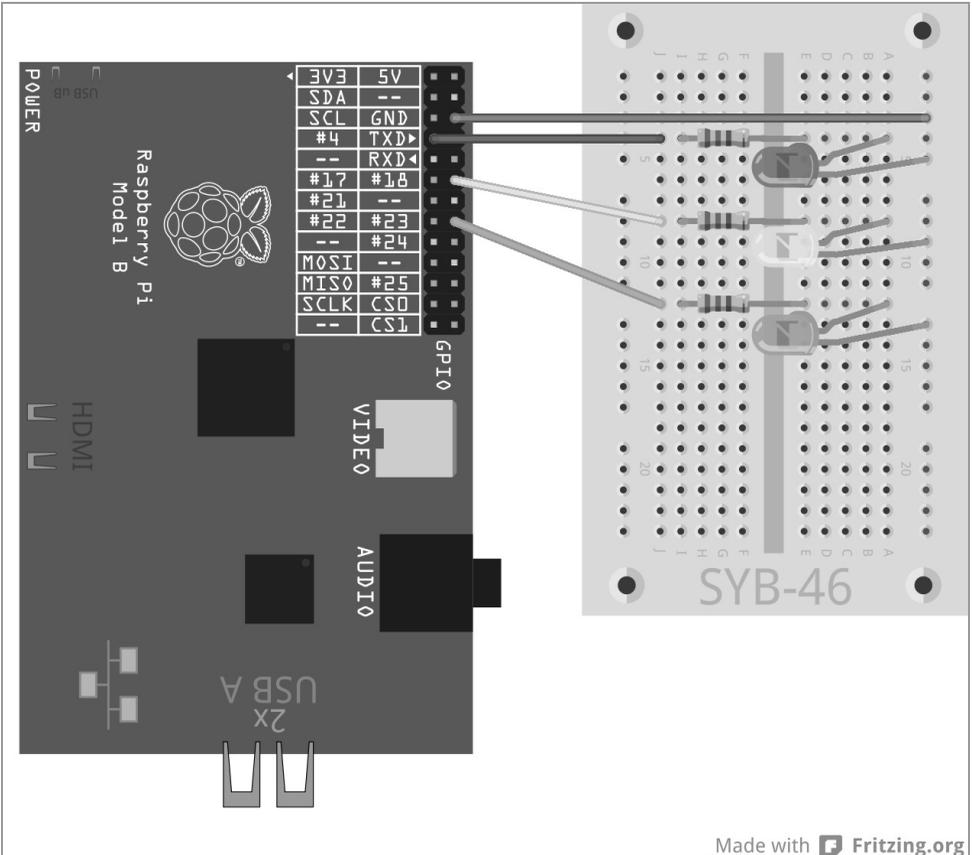
1x LED rouge

1x LED jaune

1x LED verte

3x résistances de 220 Ω

4x Câbles de connexion



Made with  Fritzing.org

Fig. 3.2: Un feu de signalisation simple

Le programme `ampe101.py` contrôle le feu de circulation :

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
rouge = 0; jaune = 1; vert = 2
Feu_de_circulation=[4,18,23]
GPIO.setup(Feu_de_circulation[rouge], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[jaune], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[vert], GPIO.OUT, initial=True)
print ("Ctrl+C arrête le programme")
try:
    while True:
        time.sleep(2)
        GPIO.output(Feu_de_circulation[vert],False);
```

```

GPIO.output(Feu_de_circulation[jaune],True)
    time.sleep(0.6)
    GPIO.output(Feu_de_circulation[jaune],False);
GPIO.output(Feu_de_circulation[rouge],True)
    time.sleep(2)
    GPIO.output(Feu_de_circulation[jaune],True)
    time.sleep(0.6)
    GPIO.output(Feu_de_circulation[rouge],False);
GPIO.output(Feu_de_circulation[jaune],False)
    GPIO.output(Feu_de_circulation[vert],True)
except KeyboardInterrupt:
    GPIO.cleanup()

```

3.1.1 Voilà comment cela fonctionne

Les premières lignes sont déjà connues. Elles importent les bibliothèques `RPi.GPIO` pour le contrôle des ports GPIO et `time` pour les retards. La numérotation des ports GPIO est ensuite définie comme dans l'exemple précédent sur BCM.

`rouge = 0; jaune = 1; vert = 2` Ces lignes définissent les trois variables `rouge`, `jaune` et `vert` pour les trois LED. Vous n'avez besoin dans le programme de rappeler aucun numéro ou de port GPIO. Vous pouvez contrôler facilement les LED via leur couleur.

`Feu_de_circulation=[4,18,23]` Pour contrôler les trois LED, une liste est établie. Elle contient les numéros GPIO dans l'ordre, dans lequel les LED sont montées sur la carte de circuit imprimé. Étant donné que les ports GPIO apparaissent uniquement à ces endroits dans le programme, vous pouvez facilement modifier le programme si vous souhaitez utiliser d'autres ports GPIO.

```

GPIO.setup(Feu_de_circulation[rouge], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[jaune], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[vert], GPIO.OUT, initial=True)

```

Les trois ports GPIO utilisés sont initialisés un par un comme des sorties. Nous n'utilisons cette fois aucun numéro de port GPIO mais plutôt la liste précédemment définie. Dans une liste, chaque élément est indexé par un numéro, en commençant par 0. `Feu_de_circulation[0]` est également le premier élément, dans ce cas 4. Les variables `rouge`, `jaune` et `vert` contiennent les numéros 0, 1 et 2, qui sont nécessaires comme index pour les éléments de la liste. De cette façon, les ports GPIO utilisés peuvent être traités par les couleurs :

- `Feu_de_circulation[rouge]` correspond au port GPIO 4 avec la LED rouge.
- `Feu_de_circulation[jaune]` correspond au port GPIO 18 avec la LED jaune.
- `Feu_de_circulation[vert]` correspond au port GPIO 23 avec la LED verte

L'instruction `GPIO.setup` peut contenir un paramètre `initial` facultatif, qui attribue déjà un statut logique au port GPIO lors de l'initialisation. Dans ce programme, nous allumons la LED verte dès le début. Les deux autres LED commencent le programme dans un état éteint.

`print("Ctrl+C arrête le programme")` Un court mode d'emploi est maintenant affiché à l'écran. Le programme s'exécute automatiquement. La combinaison de touches `[Ctrl] + [C]` doit l'arrêter. Pour vérifier si l'utilisateur a arrêté le programme avec `[Ctrl] + [C]`, nous utilisons une requête `try...except`. Le code

de programme associé sous `try` : est ensuite exécuté normalement. Si une exception système se provient pendant ce temps – cela peut être une erreur ou également la combinaison de touche `Ctrl` + `C` , le programme est interrompu et l'instruction `except` à la fin du programme est exécutée.

```
except KeyboardInterrupt:  
    GPIO.cleanup()
```

Cette combinaison de touches déclenche un `KeyboardInterrupt` et la boucle est quittée automatiquement. La dernière ligne ferme les ports GPIO utilisés et éteint toutes les LED. Le programme est terminé. Grâce à la fermeture contrôlée des ports GPIO, aucun avertissement système ou message d'erreur, qui pourrait déconcerter l'utilisateur, ne s'affiche à l'écran. Le cycle réel du feu de signalisation s'exécute dans une boucle infinie :

`while True` : Ces boucles infinies nécessitent toujours une condition d'arrêt sinon le programme ne s'arrêterait jamais.

`time.sleep(2)` Au début du programme mais également à chaque départ de la boucle, la LED verte s'allume pendant 2 secondes.

```
GPIO.output(Feu_de_circulation[vert],False);  
GPIO.output(Feu_de_circulation[jaune],True)  
time.sleep(0.6)
```

La LED verte s'éteint maintenant et la LED jaune s'allume. Elle éclaire ensuite seule pendant 0,6 seconde.

```
GPIO.output(Feu_de_circulation[jaune],False);  
GPIO.output(Feu_de_circulation[rouge],True)  
time.sleep(2)
```

La LED jaune s'éteint maintenant à nouveau et la LED rouge s'allume. Elle éclaire ensuite seule pendant 2 secondes. La phase rouge d'un feu de signalisation est généralement beaucoup plus longue que la phase jaune.

```
GPIO.output(Feu_de_circulation[jaune],True)  
time.sleep(0.6)
```

Au début de la phase rouge-jaune, la LED jaune est également allumée sans qu'une autre LED soit éteinte. Cette phase dure 0,6 seconde.

```
GPIO.output(Feu_de_circulation[rouge],False)  
GPIO.output(Feu_de_circulation[jaune],False)  
GPIO.output(Feu_de_circulation[vert],True)
```

À la fin de la boucle, le feu de signalisation repasse au vert. Les LED rouge et jaune sont éteintes et la LED verte est allumée. La boucle recommence dans la phase verte du feu de signalisation avec un temps d'attente de 2 secondes. Vous pouvez naturellement modifier tous les temps comme vous le souhaitez. En réalité, les phases du feu de signalisation dépendent des dimensions du carrefour et du trafic routier. Les phases jaune et rouge-jaune durent en général 2 secondes chacune.

4 Feu pour piétons

Dans l'expérience suivante, nous ajoutons un feu pour piéton au feu de signalisation, qui montre une lumière clignotante pour les piétons pendant la phase rouge du feu de circulation, comme cela est utilisé dans certains pays. Vous pouvez naturellement aussi installer dans le programme les feux pour piétons avec des lumières rouge et verte que l'on trouve en Europe centrale mais ce kit d'apprentissage ne contient qu'une LED en plus des LED déjà utilisées dans le feu de circulation.

Pour l'expérience suivante, montez une LED avec une pré-résistance comme montré dans le schéma du circuit. Elle doit être branchée sur le port GPIO 24.

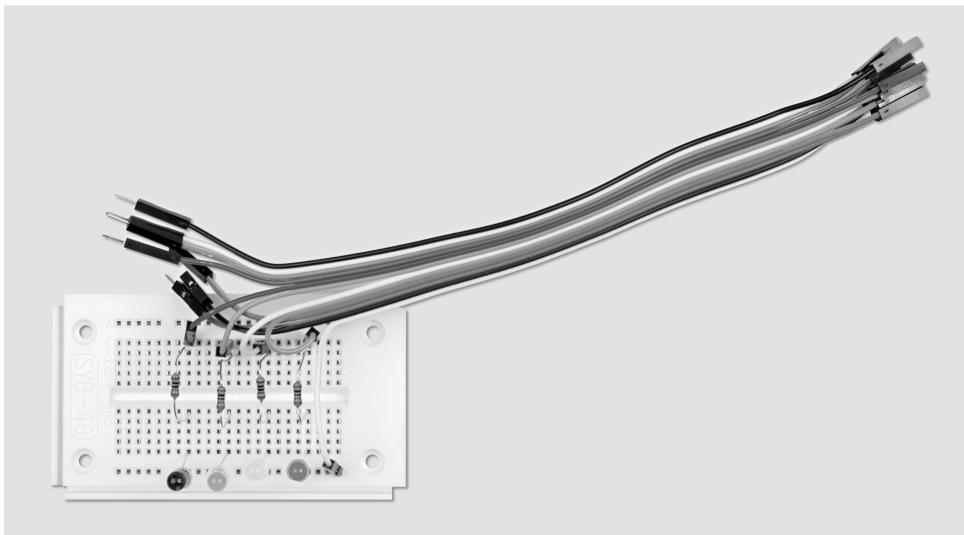


Fig. 4.1: Construction de la carte pour le feu de circulation et le feu clignotant pour piétons

Composants nécessaires :

- 1x Carte de circuit imprimé
- 1x LED rouge
- 1x LED jaune
- 1x LED verte
- 1x LED bleue
- 4x résistances de 220 Ω
- 5x câbles de connexion

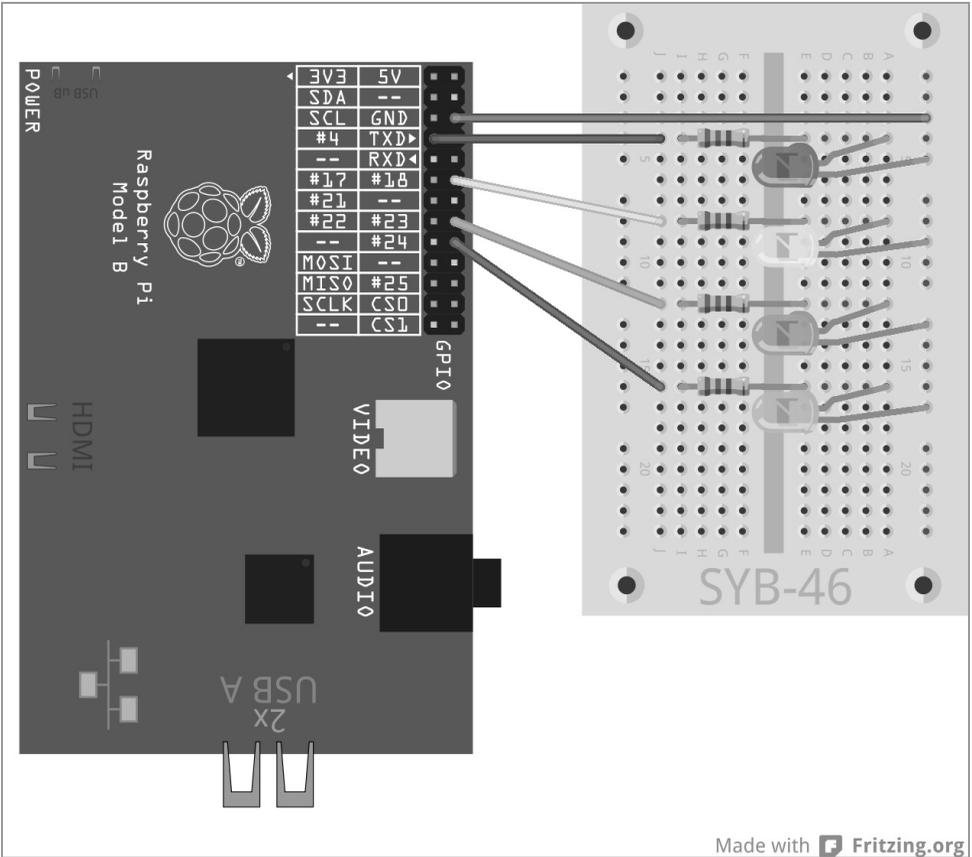


Fig. 4.2: Feu de circulation avec feu clignotant pour piéton

Le programme `ampe102.py` contrôle le nouveau système de feu de signalisation : Le programme a été légèrement développé par rapport à la version précédente.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
rouge = 0; jaune = 1; vert = 2, bleu = 3
Feu_de_circulation=[4,18,23,24]
GPIO.setup(Feu_de_circulation[rouge], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[jaune], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[vert], GPIO.OUT, initial=True)
GPIO.setup(Feu_de_circulation[bleu], GPIO.OUT, initial=False)
print ("Ctrl+C arrête le programme")
try:
    while True:
```

```

        time.sleep(2)
        GPIO.output(Feu_de_circulation[vert],False);
GPIO.output(Feu_de_circulation[jaune],True)
        time.sleep(0.6)
        GPIO.output(Feu_de_circulation[jaune],False);
GPIO.output(Feu_de_circulation[rouge],True)
        time.sleep(0.6)
        for i in range(10):
            GPIO.output(Feu_de_circulation[bleu],True); time.sleep(0.05)
            GPIO.output(Feu_de_circulation[bleu],False); time.sleep(0.05)
        time.sleep(0.6)
        GPIO.output(Feu_de_circulation[jaune],True); time.sleep(0.6)
        GPIO.output(Feu_de_circulation[rouge],False)
        GPIO.output(Feu_de_circulation[jaune],False)
        GPIO.output(Feu_de_circulation[vert],True)
except KeyboardInterrupt:
    GPIO.cleanup()

```

4.1.1 Voilà comment cela fonctionne

Le déroulement du programme est déjà connu. Pendant la phase rouge longue, le feu pour piétons bleu doit clignoter rapidement.

`bleu = 4` Une nouvelle variable définit la LED pour le feu pour piéton dans la liste

`Feu_de_circulation=[4,18,23,24]` La liste a été élargie à quatre éléments pour pouvoir contrôler les quatre LED.

`GPIO.setup(Feu_de_circulation[bleu], GPIO.OUT, initial=False)` La nouvelle LED est initialisée et est éteinte au début. C'est le réglage de base pendant la phase verte du feu de circulation.

```

        time.sleep(0.6)
        for i in range(10):
            GPIO.output(Feu_de_circulation[bleu],True); time.sleep(0.05)
            GPIO.output(Feu_de_circulation[bleu],False); time.sleep(0.05)
        time.sleep(0.6)

```

Dans le cycle du feu de signalisation, une boucle qui permet à la LED bleue de clignoter, commence 0,6 seconde après le début de la phase rouge. Pour ce faire, nous utilisons ici une boucle `for` qui, contrairement à la boucle `while` sera répétée un nombre défini de cycles de boucle, et qui ne s'exécute pas tant qu'une certaine condition d'arrêt n'est pas satisfaite.

`for i in range(10):` Chaque boucle `for` nécessite un compteur de boucle, une variable qui change de valeur à chaque répétition de boucle. Pour un compteur de boucle simple, le nom de variable `i` s'est établi dans tous les langages de programmation. Tout autre nom est naturellement possible. Cette valeur peut être appelée comme toute autre variable dans la boucle, mais ce n'est pas nécessaire ici. Le paramètre `range` dans la boucle indique le nombre de cycle de la boucle, et plus précisément, quelles sont les valeurs que le compteur de boucle peut donner. Dans notre exemple, la boucle est répétée 10 fois. Le compteur de boucle `i` passe de la valeur 0 à 9. Dans la boucle, la nouvelle LED bleue est allumée puis éteinte après 0,05 seconde. Un cycle de boucle est terminé après 0,05 seconde supplémentaire, et le prochain cycle recommence avec l'allumage de la LED. De cette façon, elle clignote dix fois, et la phase complète dure 1 seconde.

`time.sleep(0.6)` Avec un retard de 0,6 seconde après le dernier cycle de boucle, le cycle de commutation normal du feu de circulation est poursuivi par l'allumage simultané de la LED jaune et de la LED rouge (déjà allumée). Rien de nouveau jusqu'à maintenant. Ce qui est vraiment intéressant, c'est le feu pour piéton, lorsqu'il ne fonctionne pas automatiquement mais qu'il doit d'abord être activé en appuyant sur un bouton, comme c'est le cas pour de nombreux feux pour piétons. Dans la prochaine expérience, un bouton connecté à un port GPIO simulera le bouton-poussoir que l'on trouve sur un vrai feu pour piéton.

4.2 Bouton sur la connexion GPIO

Les ports GPIO peuvent non seulement être utilisés pour la sortie de données, par exemple via les LED, mais également être utilisés pour l'entrée de données. Pour ce faire, vous devez les définir comme des entrées dans le programme. Pour les entrées, nous utilisons dans le projet suivant un bouton qui est directement inséré dans la carte de circuit imprimé. Le bouton a quatre broches de connexion et chaque paire de broches opposées (les plus distantes) est interconnectée. Tant que vous appuyez sur le bouton, les quatre connexions sont interconnectées. Contrairement à un interrupteur, un bouton ne se verrouille pas. La connexion est interrompue immédiatement dès que vous relâchez le bouton. Si un signal de + 3,3 V est délivré sur le port GPIO défini comme entrée, ce dernier est considéré comme un `True` ou `1` logique. Vous pouvez théoriquement connecter le port GPIO avec la connexion + 3,3 V du Raspberry Pi via le bouton. Cependant, vous ne devez le faire en aucun cas ! Cela surchargerait le port GPIO. Connectez toujours une résistance de protection de 1 k Ω entre une entrée GPIO et la connexion de + 3,3 V pour éviter que trop de courant n'arrive sur le port GPIO et par conséquent sur le processeur.

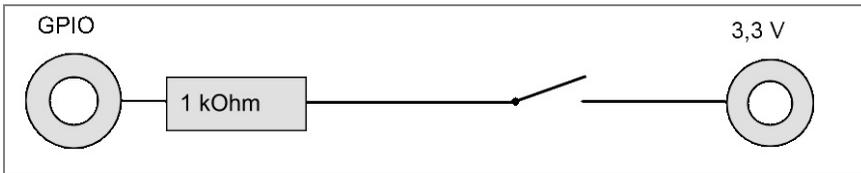


Fig. 4.3: Touche avec résistance de protection sur une entrée GPIO

Dans la plupart des cas, ce circuit simple fonctionne déjà. Cependant, le port GPIO n'a pas d'état clairement défini lorsque le bouton est ouvert. Si un programme appelle ce port, il est possible d'obtenir des résultats aléatoires. Pour éviter cela, connectez une résistance relativement très élevée - généralement 10 k Ω - à la masse. Cette résistance dite Pull-down tire le statut du port GPIO vers le bas à 0 V lorsque le bouton est ouvert. Étant donné que la résistance est très élevée, il n'y a pas de risque de court-circuit tant que vous appuyez sur le bouton. Lorsque vous appuyez sur le bouton, + 3,3 V et le fil de terre sont directement connectés via cette résistance.

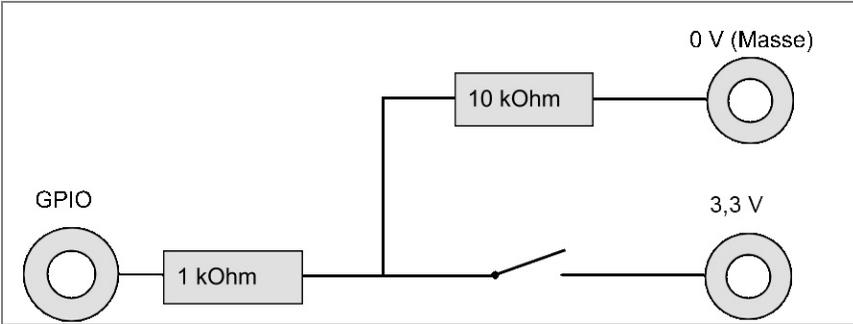


Fig. 4.4: Touche avec résistance de protection et résistance Pull-Down sur une entrée GPIO

Montez un bouton selon la figure suivante en utilisant les deux résistances dans le circuit.

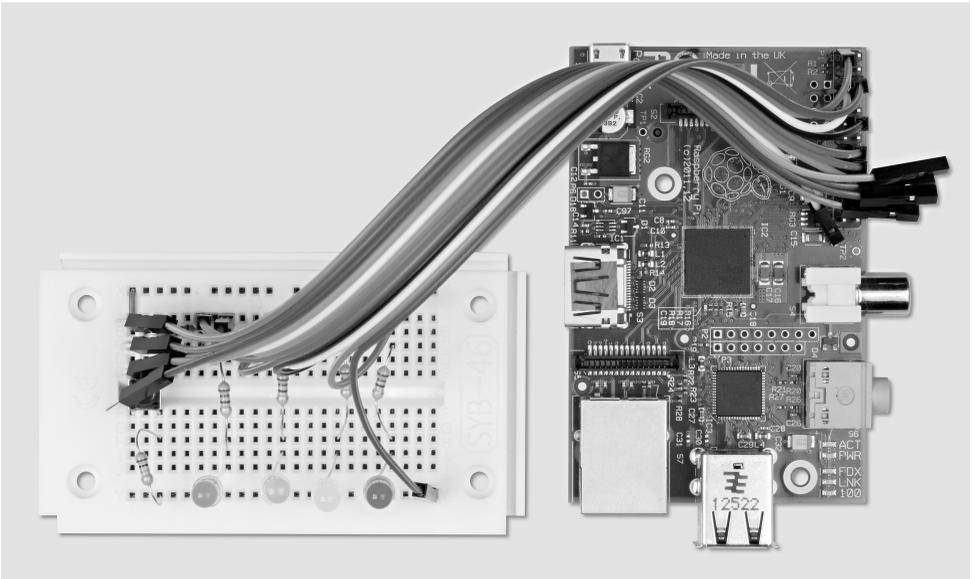


Fig. 4.5: Construction de la carte pour le feu pour piéton « activé sur demande ».

Composants nécessaires :

1x Carte de circuit imprimé

1x LED rouge

1x LED jaune

1x LED verte

1x LED bleue

4x résistances de 220 Ω

1x Bouton

1x résistance de 1 k Ω

1x résistance de 10 k Ω

7x câbles de connexion

1x cavalier court

La barre de contact en bas de la figure du bouton est connectée avec le fil + 3,3 V du Raspberry Pi (broche 1) via le rail plus de la carte de circuit imprimé. Pour connecter le bouton avec le rail plus, nous utilisons un cavalier court pour garder le dessin clair. Sinon, vous pouvez également connecter un des contacts inférieurs du bouton directement avec la broche 1 du Raspberry Pi en utilisant un câble de connexion.

La barre de contact supérieure dans la figure du bouton est connectée au port GPIO 25 via une résistance de protection de 1 k Ω (Marron-Noir-Rouge) et au fil de terre via une résistance Pull-Down de 10 k Ω (Marron-Noir-Orange).

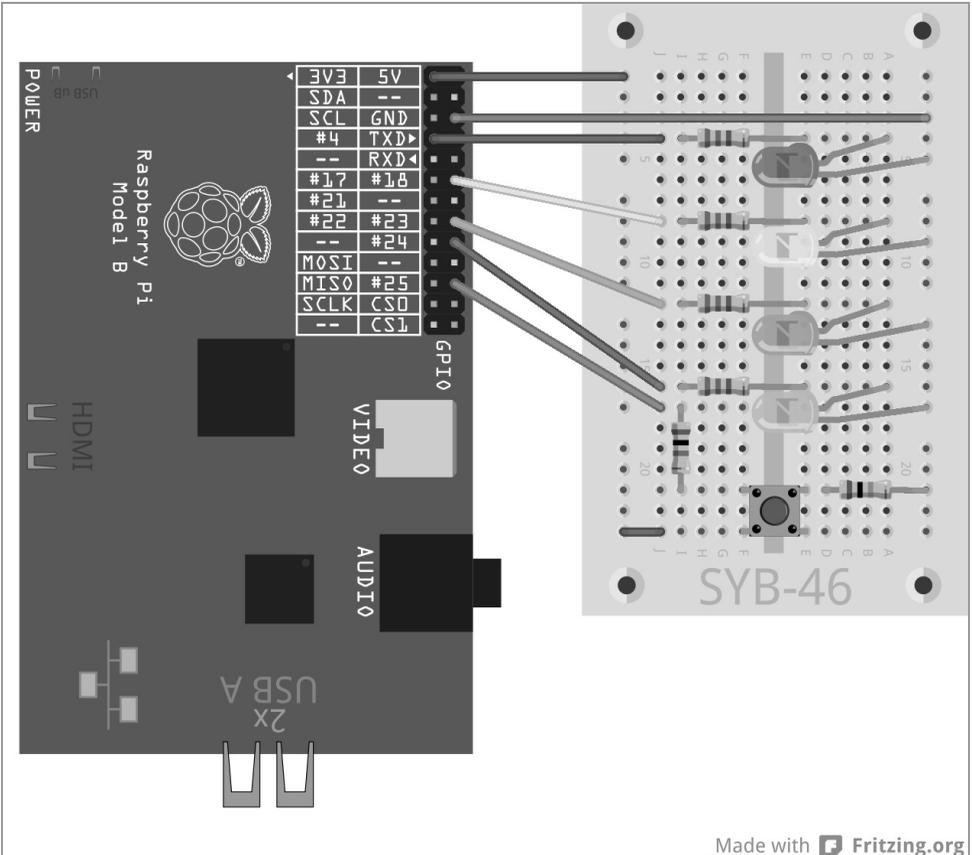


Fig. 4.6: Feu clignotant pour piéton avec bouton

Le programme `ampe103.py` contrôle le nouveau système de feu de signalisation avec le bouton pour le feu clignotant pour piéton

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

rouge    = 0; jaune  = 1; vert  = 2; bleu  = 3; bouton = 4

Feu_de_circulation=[4,18,23,24,25]
GPIO.setup(Feu_de_circulation[rouge], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[jaune], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[vert], GPIO.OUT, initial=True)
```

```

GPIO.setup(Feu_de_circulation[bleu], GPIO.OUT, initial=False)
GPIO.setup(Feu_de_circulation[bouton], GPIO.IN)

print ("Appuyer sur le bouton pour activer le feu clignotant pour piétons
, Ctrl+C arrête le programme")

try:
    while True:
        if GPIO.input(Feu_de_circulation[bouton])==True:
            GPIO.output(Feu_de_circulation[vert],False)
            GPIO.output(Feu_de_circulation[jaune],True)
            time.sleep(0.6)
            GPIO.output(Feu_de_circulation[jaune],False)
            GPIO.output(Feu_de_circulation[rouge],True)
            time.sleep(0.6)
            for i in range(10):
                GPIO.output(Feu_de_circulation[bleu],True); time.sleep(0.05)
                GPIO.output(Feu_de_circulation[bleu],False); time.sleep(0.05)
            time.sleep(0.6)
            GPIO.output(Feu_de_circulation[jaune],True)
            time.sleep(0.6)
            GPIO.output(Feu_de_circulation[rouge],False);
GPIO.output(Feu_de_circulation[jaune],False)
            GPIO.output(Feu_de_circulation[vert],True); time.sleep(2)
except KeyboardInterrupt:
    GPIO.cleanup()

```

4.2.1 Voilà comment cela fonctionne

Le programme est légèrement développé par rapport à la dernière version.

-*- coding: utf-8 -*- Afin d'afficher correctement les [accents français] du feu clignotant pour piétons dans la sortie de programme - indépendamment des paramètres de l'interface IDLE définis par l'utilisateur, un code pour afficher les caractères spéciaux est défini dès le début. Cette ligne doit être incluse dans tous les programmes qui affichent des textes, dans lesquels il y a des [accents] ou d'autres caractères spéciaux propres à chaque pays.

ASCII, ANSI et Unicode

Un alphabet normal contient 26 lettres plus quelques caractères spéciaux, tous en majuscules et en minuscules, ainsi que 10 chiffres et quelques signes de ponctuation ; l'ensemble s'élève à environ 100 signes différents. Un octet permet d'afficher 256 signes différents. Cela devrait être suffisant - pensez au début de l'histoire des ordinateurs, lorsque les bases importantes des techniques actuelles ont été définies.

Il est apparu très rapidement que l'inventeur du jeu de caractère ASCII (American Standard Code for Information Interchange) basé sur 256 signes s'était trompé. C'était un américain qui n'avait pas pensé au-delà du monde anglophone. Dans toutes les langues importantes dans le monde, sans compter les langues asiatiques et arabes avec leur propre système d'écriture, il existe des centaines de lettres qui doivent être représentées. Elles ne peuvent pas toutes tenir dans l'espace libre disponible de la liste étendue des 256 caractères.

Plus tard, le jeu de caractères ANSI a été introduit parallèlement au jeu de caractère ASCII. Il est utilisé par les anciennes versions de Windows et présente encore une fois les mêmes erreurs. Pour parfaire la confusion des langues, les [accents français] et les autres lettres avec des accents ont été classés dans un autre endroit du jeu de caractères dans la norme ASCII.

Pour résoudre ce problème, l'Unicode a été introduit dans les années 1990. Il représente toutes les langues, y compris les hiéroglyphes égyptiens, l'écriture cunéiforme et le sanskrit védique, l'écriture la plus ancienne dans le monde. Pour coder les caractères Unicode de la forme la plus commune dans les fichiers de texte bruts, il y a UTF-8 un code qui fonctionne sur plusieurs plateformes, qui correspond aux 128 premiers caractères de l'ASCII et qui est rétrocompatible avec quasiment tous les systèmes affichant des textes. Le codage est représenté dans une ligne de commentaire. Toutes les lignes qui commencent avec un signe #, ne sont pas prise en compte par l'interpréteur Python. Le codage, qui doit toujours figurer au début d'un programme, informe le Python-Shell des caractères à afficher sans qu'il y ait de réelle instruction de programmation. De cette façon, vous pouvez également entrer vos commentaires dans le code du programme.

Commentaires dans le programme

Lorsque l'on écrit un programme, il est possible que l'on ne se souvienne pas plus tard ce que l'on avait en tête pour certaines instructions du programme. La programmation est l'une des activités les plus créatives car l'on crée quelque chose uniquement et exclusivement à partir de ses idées sans limite posée par le matériel et les outils. Les commentaires sont particulièrement important lorsque l'on programme, pour qu'une autre personne comprenne ce que l'on fait ou pour modifier le programme. Aucun commentaire n'est inclus dans les exemples de programme pour maintenir un code de programme clair. Toutes les instructions du programme sont décrites dans le détail.

Lorsque l'on publie son propre programme, il faut toujours se poser la question : les commentaires en [français] ou en anglais ? En cas de commentaires en [français], les allemands se plaignent de ne pas comprendre la langue. Avec des commentaires en anglais, on se comprend parfois plus et les Anglais se moquent du mauvais anglais utilisé.

```
bouton = 4
Feu_de_circulation=[4,18,23,24,25]
```

Par souci de simplicité, pour le bouton, un seul élément supplémentaire avec le numéro 4 et le port GPIO 25 a été inséré dans la liste. De cette façon, vous pouvez également choisir un autre port GPIO pour le bouton, dont le numéro comme le port GPIO de la LED est inscrit seulement à cet endroit du programme.

`GPIO.setup(Feu_de_circulation[bouton], GPIO.IN)` Le port GPIO du bouton est défini comme une entrée. Cette définition s'effectue également via `GPIO.setup`, mais cette fois avec le paramètre `GPIO.IN`.

```
print ("Appuyer sur le bouton pour activer le feu clignotant pour piétons, Ctrl+C
arrête le programme")
```

Lors du démarrage, le programme affiche un long message qui informe l'utilisateur qu'il doit appuyer sur le bouton.

```
while True:
    if GPIO.input(Feu_de_circulation[bouton])==True:
```

Dans la boucle infinie, une requête est maintenant intégrée. Les instructions suivantes sont d'abord exécutées lorsque le port GPIO 25 donne la valeur `True` et que l'utilisateur appuie également sur le bouton. Jusqu'à ce moment, le feu de circulation reste dans sa phase verte. La poursuite de la boucle correspond essentiellement au déroulement du dernier programme. Le feu de circulation passe du jaune au rouge et la lumière clignotante clignote dix fois. Ensuite, le feu de circulation repasse au rouge-jaune, puis au vert.

`time.sleep(2)` Il y a une petite différence dans ce programme. La phase verte de 2 secondes est intégrée maintenant à la fin de la boucle et n'est plus au début de la boucle. Elle est néanmoins appliquée une fois par cycle de boucle, avec comme différence que le cycle du feu de circulation commence immédiatement et sans retard lorsque vous appuyez sur le bouton. Pour éviter que la phase verte n'échoue lorsque vous réappuyez sur le bouton immédiatement après la phase jaune, ce retard est maintenant intégré à la fin de la boucle.

5 Motif à LED coloré et chenillard

Les chenillards sont toujours des effets populaires pour attirer l'attention, que ce soit dans une salle de jeux ou dans les enseignes lumineuses professionnelles. Grâce au Raspberry Pi et à quelques LED, vous pouvez réaliser facilement quelque chose de ressemblant.

Montez pour l'expérience suivante quatre LED avec des pré-résistances comme indiqué sur la figure. Ce circuit correspond au feu pour piétons sans le bouton de l'expérience précédente.

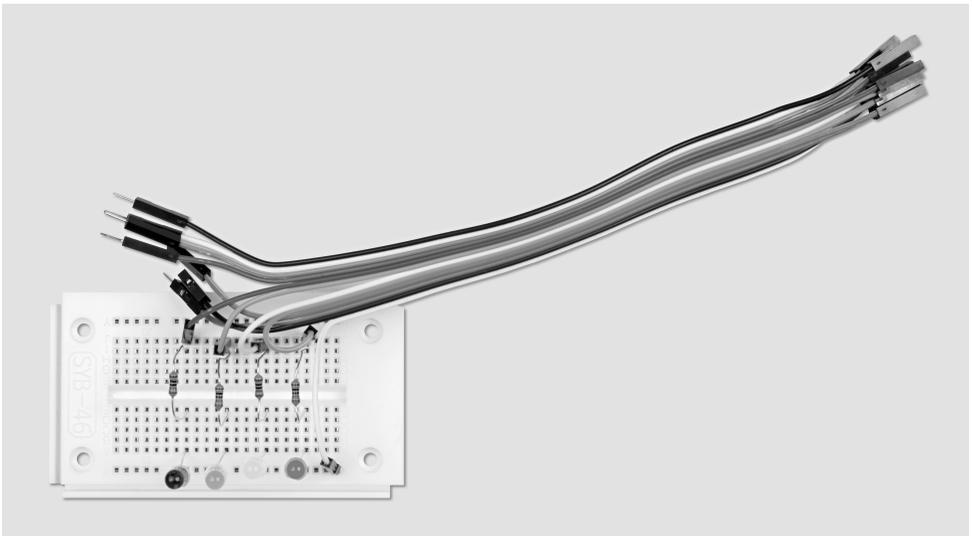


Fig. 5.1: Construction de carte pour le motif et le chenillard.

- Composants nécessaires :
- 1x Carte de circuit imprimé
 - 1x LED rouge
 - 1x LED jaune
 - 1x LED verte
 - 1x LED bleue
 - 4x résistances de 220 Ω
 - 5x câbles de connexion

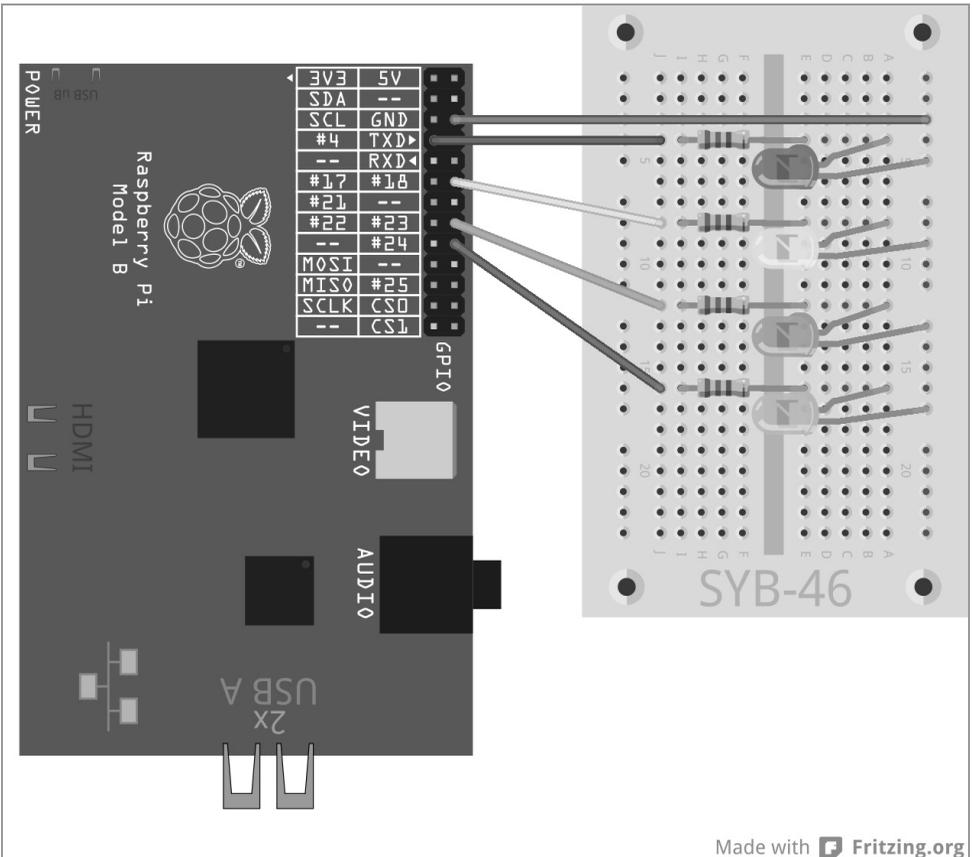


Fig. 5.2: Quatre LED avec pré-résistances.

Grâce à différents motifs clignotants à LED, nous expliquons plusieurs boucles et méthodes de programmation en Python. Le programme suivant propose différents motifs à LED qui peuvent être choisis par l'utilisateur en utilisant le clavier.

Le programme `ledmuster.py` permet de faire clignoter les LED de différentes façons.

```
# -*- coding: utf-8 -*-
import RPi.GPIO as GPIO
import time
import random

GPIO.setmode(GPIO.BCM)

LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)

z = len(LED); w = 5; t = 0.2

print ("Choix de l'effet lumineux"); print ("1 - chenillard cyclique")
print ("2 - chenillard aller-retour"); print ("3 - montées et descentes")
print ("4 - Tous clignotent simultanément"); print ("5 - tous clignotent
aléatoirement")
print ("Ctrl+C arrête le programme")

try:
    while True:
        e = raw_input ("Veuillez choisir le motif : ")
        if e == "1":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
        elif e == "2":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
                for j in range(z-1, -1, -1):
                    GPIO.output(LED[j], True); time.sleep(t)
                    GPIO.output(LED[j], False)
        elif e == "3":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True); time.sleep(t)
                    time.sleep(2*t)
                for j in range(z-1, -1, -1):
                    GPIO.output(LED[j], False)
                    time.sleep(t)
                    time.sleep(2*t)
        elif e == "4":
            for i in range(w):
                for j in range(z):
                    GPIO.output(LED[j], True)
                    time.sleep(2*t)
                for j in range(z):
                    GPIO.output(LED[j], False)
                    time.sleep(t)
        elif e == "5":
```

```

        for i in range(w*z):
            j = random.randint(0,z-1)
            GPIO.output(LED[j], True); time.sleep(t)
            GPIO.output(LED[j], False)
        else:
            print ("Entrée invalide")

except KeyboardInterrupt:
    GPIO.cleanup()

```

5.1.1 Voilà comment cela fonctionne

La première ligne du programme avec la définition du codage UTF-8 et l'importation des bibliothèques requises sont déjà connues grâce aux expériences précédentes. La bibliothèque `random` est importée en plus pour générer un motif clignotant aléatoire.

Lorsque ce programme a été développé, on a veillé à ce qu'il soit possible de l'utiliser de différentes façons. Il peut donc être utilisé sans problème avec plus de quatre LED. Cette flexibilité fait aujourd'hui partie d'un bon style de programmation. En prenant l'exemple du Raspberry Pi, ce type de programmes permet non seulement de s'étendre vers de nouveaux ports GPIO, mais également de modifier facilement d'autres ports GPIO lorsqu'ils sont nécessaires au niveau matériel.

`LED = [4, 18, 23, 24]` Pour les LED, une liste avec les numéros de port GPIO est à nouveau définie pour que ces ports soient établis seulement à un endroit dans le programme.

```

for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)

```

Au lieu d'initialiser les ports GPIO de chaque LED comme dans les programmes précédents, une boucle `for` s'exécute cette fois pour la liste `LED`. Le compteur de boucle `i` prend successivement chaque valeur dans la liste, p. ex les numéros de port GPIO des LED, et n'est pas simplement incrémenté comme dans les boucles `for` précédemment utilisées. De longues listes de votre choix peuvent être traitées de cette façon. La longueur de la liste doit être ignorée lors du développement du programme.

Les quatre ports GPIO pour les LED sont définis comme des sorties et mis sur 0 pour éteindre les LED éventuellement allumées lors des expériences précédentes.

```

z = len(LED); w = 5; t = 0.2

```

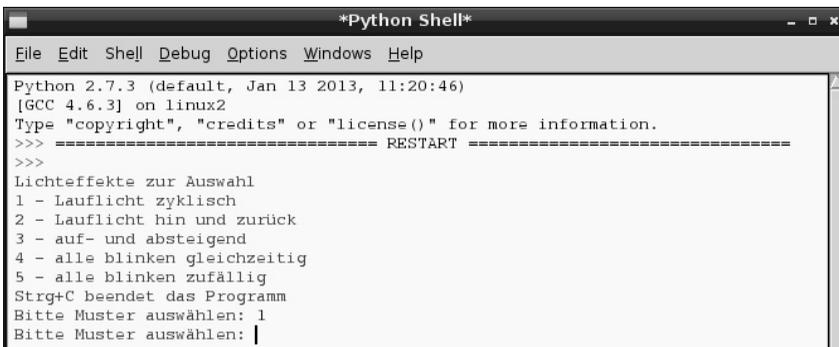
Pour garder le programme universel et facile à modifier, trois autres variables sont maintenant définies :

z	Nombre de LED	Le nombre de LED est automatiquement repris à l'aide de la fonction <code>len()</code> à partir de la liste <code>LED</code> .
w	Répétitions	Chaque motif doit être répété cinq fois par défaut pour être mieux reconnu. Ce nombre peut être modifié selon vos souhaits et s'applique ensuite à tous les motifs.
t	Temps	Cette variable indique la durée de l'allumage de la LED lorsqu'elle clignote. La pause qui suit est aussi longue. Le nom <code>t</code> s'est établi pour les variables qui enregistrent les périodes de temps dans les programmes, dans presque tous les langages de programmation.

Les valeurs définies comme variables sont non seulement intégrées de manière fixe à cet endroit dans le programme mais peuvent également être facilement modifiées. Le vrai programme commence après ces définitions.

```
print ("Choix de l'effet lumineux"); print ("1 - chenillard cyclique")
print ("2 - chenillard aller-retour"); print ("3 - montées et descentes")
print ("4 - Tous clignent simultanément"); print ("5 - tous clignent
aléatoirement")
print ("Ctrl+C arrête le programme")
```

Ces lignes affichent à l'écran des instructions pour l'utilisateur pour la correspondance des touches numériques et des différents motifs présentés.



```
*Python Shell*
File Edit Shell Debug Options Windows Help
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Lichteffekte zur Auswahl
1 - Lauflicht zyklisch
2 - Lauflicht hin und zurück
3 - auf- und absteigend
4 - alle blinken gleichzeitig
5 - alle blinken zufällig
Strg+C beendet das Programm
Bitte Muster auswählen: 1
Bitte Muster auswählen: |
```

Fig. 5.3: Le programme sur l'écran

Après que la sélection a été affichée, la boucle principale du programme se lance. Pour ce faire, nous utilisons ici également une boucle infinie `while True:` qui est intégrée dans une instruction `try...except`.

`e = raw_input("Veuillez choisir le motif : ")` Dès le début de la boucle, le programme attend une entrée de l'utilisateur, qui est enregistrée dans la variable `e`. La fonction `raw_input()` prend l'entrée dans le texte en clair sans l'interpréter. En revanche, les opérations mathématiques entrées avec `input()` ou

les noms de variables sont directement interprétés. Dans la plupart de cas, `raw_input()` est également le meilleur choix car vous n'avez pas besoin de vous occuper des nombreuses entrées possibles.

Le programme attend jusqu'à ce que l'utilisateur entre une lettre et appuie sur le bouton `Enter`. Selon le chiffre que l'utilisateur a entré, un motif donné doit maintenant être affiché avec les LED. Pour le vérifier, nous utilisons une construction `if...elif...else`.

Motif 1

Si l'entrée était un 1, la partie du programme indentée derrière cette ligne est exécutée.

`if e == "1"`: Notez que l'indentation dans Python ne sert pas seulement d'aide visuelle. Comme avec les boucles, les requêtes sont également initiées avec une indentation.

Égal n'est pas égal égal

Python utilise deux types de signe pour l'égalité. Le signe `=` simple est utilisé pour attribuer une valeur donnée à une variable. Le double signe égal `==` est utilisé dans les requêtes et vérifie si deux valeurs sont vraiment identiques.

Si l'utilisateur a également entré un 1 avec le clavier, une boucle est lancée, qui génère un chenillard cyclique. Ces boucles sont construites en principe de la même façon pour tous les motifs à LED utilisés.

`for i in range(w)`: La boucle externe répète le motif aussi souvent que spécifié dans les variables `w` définies dans l'entrée. Il y a une autre boucle dans cette boucle, qui génère chaque motif. Elle diffère avec chaque motif.

```
for j in range(z):
    GPIO.output(LED[j], True); time.sleep(t)
    GPIO.output(LED[j], False)
```

Dans le cas d'un simple chenillard cyclique, cette boucle s'exécute une fois pour chaque LED de la liste, les unes après les autres. Le nombre de LED est enregistré au début du programme dans la variable `z`. La LED avec le numéro correspondant à l'état actuel du compteur de cycle est allumée. Ensuite, le programme attend l'entrée pendant le temps enregistré dans la variable `t` puis éteint à nouveau la LED. Le cycle suivant de boucle commence ensuite avec la LED suivante. La boucle externe répète la même boucle interne cinq fois.

Motif 2

Si l'utilisateur a entré un 2, une boucle semblable est lancée. Ici, les LED sont non seulement comptées dans un sens mais également dans le sens inverse à la fin du chenillard. La lumière avance et recule en alternance.

`elif e == "2"`: Les autres requêtes après la première utilisent la requête `elif`, ce qui signifie qu'elles ne peuvent être exécutées que si la requête précédente a retourné un résultat `False`.

```

for i in range(w):
    for j in range(z):
        GPIO.output(LED[j], True); time.sleep(t)
        GPIO.output(LED[j], False)
    for j in range(z-1, -1, -1):
        GPIO.output(LED[j], True); time.sleep(t)
        GPIO.output(LED[j], False)

```

Des boucles imbriquées les unes dans les autres sont également utilisées ici. Après la première boucle interne, qui correspond à la partie du programme décrite ci-dessus, après que la LED avec le numéro 3 s'est allumée, une autre boucle pour le chenillard allant à contresens commence. Les éléments de la liste sont toujours numérotés à partir de 0. La quatrième LED a donc le numéro 3.

Pour qu'une boucle puisse s'exécuter en arrière, nous utilisons la syntaxe étendue de `for...range()`. A la place de spécifier seulement une valeur finale, trois paramètres peuvent également être définis : la valeur initiale, l'incrément et la valeur finale. Dans notre exemple, ils étaient :

Valeur initiale	z-1	La variable <i>z</i> contient le nombre de LED. Étant donné que la numérotation des éléments de la liste commence avec 0, la dernière LED a le numéro <i>z</i> -1.
Incrément	-1	Avec un incrément égal à -1. Chaque cycle de boucle décompte un nombre en arrière.
Valeur finale	-1	La valeur finale dans une boucle est toujours la première valeur qui n'est pas atteinte. Dans la première boucle comptée en avant, le compteur de boucle commence à 0 et atteint la valeur 0, 1, 2, 3 dans notre exemple pour adresser les LED. Le 4 n'est pas atteint en quatre cycles de boucles. La boucle décomptée en arrière doit finir avec 0 et la valeur -1 ne doit donc pas être atteinte.

La deuxième boucle décrite permet de faire clignoter les quatre LED successivement dans le sens inverse. Après cela, la boucle externe recommence le cycle dans son ensemble qui dure ici deux fois plus longtemps que la première partie du programme parce que chaque LED clignote deux fois.

Motif 3

Si l'utilisateur a entré un 3, une boucle semblable est lancée. Ici, les LED sont également comptées dans les deux sens mais pas immédiatement après l'allumage puis l'extinction.

```

elif e == "3":
    for i in range(w):
        for j in range(z):
            GPIO.output(LED[j], True); time.sleep(t)
            time.sleep(2*t)
        for j in range(z-1, -1, -1):
            GPIO.output(LED[j], False); time.sleep(t)
            time.sleep(2*t)

```

La première boucle interne allume les LED les unes après les autres avec un retard. À la fin de la boucle qui est reconnaissable par l'indentation de la ligne `time.sleep(2*t)`, on attend pendant deux fois le temps de retard. Pendant ce temps, toutes les LED sont allumées. Après cela, une nouvelle boucle commence, qui décompte en arrière et qui éteint les LED, une par une. On attend également à la fin deux fois le temps de retard, lorsque toutes les LED sont éteintes, avant que la boucle externe recommence encore une fois le cycle en entier.

Motif 4

Si l'utilisateur a entré un 4, un autre motif clignotant commence, avec lequel toutes les LED clignotent en même temps et ne s'allument pas les unes après les autres.

```
elif e == "4":
    for i in range(w):
        for j in range(z):
            GPIO.output(LED[j], True)
            time.sleep(2*t)
        for j in range(z):
            GPIO.output(LED[j], False)
            time.sleep(t)
```

Étant donné que plusieurs ports GPIO ne peuvent pas être activés ou inactivés avec une seule instruction, des boucles sont également utilisées ici, cependant sans retard dans la boucle. Les quatre LED sont immédiatement allumées les unes après les autres. Cela semble au même moment pour l'œil humain. À la fin de la première boucle interne, le programme attend deux fois le temps de retard avant que toutes les LED sont éteintes à nouveau.

Les différents temps d'éclairage et d'extinction permettent de créer différents effets avec les lumières clignotantes. Le clignotement est plutôt perçu quand le temps d'éclairage est plus long que le temps d'extinction. Des temps d'éclairage très courts associés avec des temps d'extinction relativement longs génèrent un effet de flash.

Motif 5

Si l'utilisateur a entré un 5, les LED clignotent complètement aléatoirement.

```
elif e == "5":
    for i in range(w*z):
        j = random.randint(0,z-1)
        GPIO.output(LED[j], True); time.sleep(t)
        GPIO.output(LED[j], False)
```

Comme aucune boucle imbriquée n'est utilisée ici, nous laissons la boucle se répéter plus souvent. Grâce à la multiplication des variables `w` et `z`, chaque LED clignote en moyenne autant que dans le premier motif.

La fonction `random.randint()` écrit un chiffre aléatoire dans la variable `j`. Ce chiffre aléatoire est supérieur ou égal au premier paramètre et inférieur ou égal au deuxième paramètre. Dans notre cas la valeur peut également être 0, 1, 2, 3.

La LED sélectionnée aléatoirement est allumée puis éteinte à nouveau après un temps de retard. Après cela, la boucle redémarre et une nouvelle LED est choisie aléatoirement.

Entrée invalide

Pour tous les programmes qui nécessitent une entrée de l'utilisateur, on doit intercepter les entrées incorrectes. Si l'utilisateur entre une réponse imprévue, le programme doit y répondre.

```
else:  
    print ("Entrée invalide")
```

Si l'utilisateur a entré quelque chose d'autre, l'instruction donnée sous `else` est exécutée. On rencontre toujours cette partie de requête si aucune des autres requêtes ne fournit de véritable résultat. Dans notre cas, le programme affiche un message à l'écran.

Comme dans les expériences précédentes, le programme est quitté avec un `KeyboardInterrupt`, dans lequel l'utilisateur appuie sur la combinaison de touches `Ctrl` + `C`. La dernière ligne ferme les ports GPIO utilisés et éteint toutes les LED.

6 Variation de l'intensité lumineuse des LED par modulation de la largeur des impulsions

Les LED sont des composants typiques pour émettre des signaux dans l'électronique numérique. Elles peuvent être dans deux états différents, marche et arrêt, 0 et 1 ou `True` et `False`. Il en est de même pour les ports GPIO définis comme des sorties numériques. Par conséquent, il n'est théoriquement pas possible de faire varier l'intensité lumineuse d'une LED.

Une astuce permet cependant de réguler la luminosité d'une LED sur un port GPIO numérique. Si l'on fait clignoter assez rapidement une LED, l'œil humain ne l'interprète pas comme un clignotement. La technique désignée comme la modulation de la largeur des impulsions génère un signal pulsé qui s'active et se désactive à des intervalles très courts. La tension du signal reste toujours constante, seul le rapport entre le niveau `False` (0 V) et le niveau `True` (+ 3,3 V) est changé. Le rapport cyclique donne le rapport de la longueur de l'état activé sur la durée totale d'un cycle de commutation.

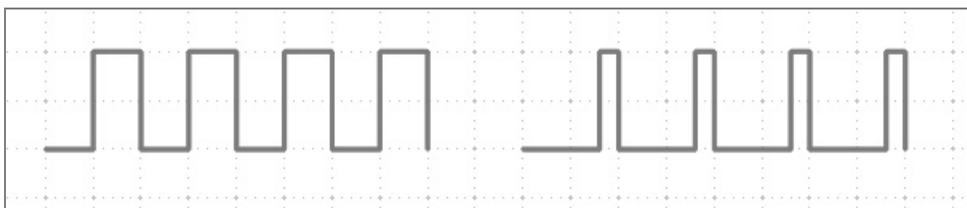


Fig. 6.1: A gauche : Rapport cyclique : 50 % - à droite : Rapport cyclique : 20 %

Plus le rapport cyclique est cyclique, plus le temps d'éclairage de la LED est court dans un cycle de commutation. De cette façon, une LED semble sombre comme une LED éteinte en permanence.

Pour l'expérience suivante, branchez une LED via une pré-résistance sur le port GPIO 18.

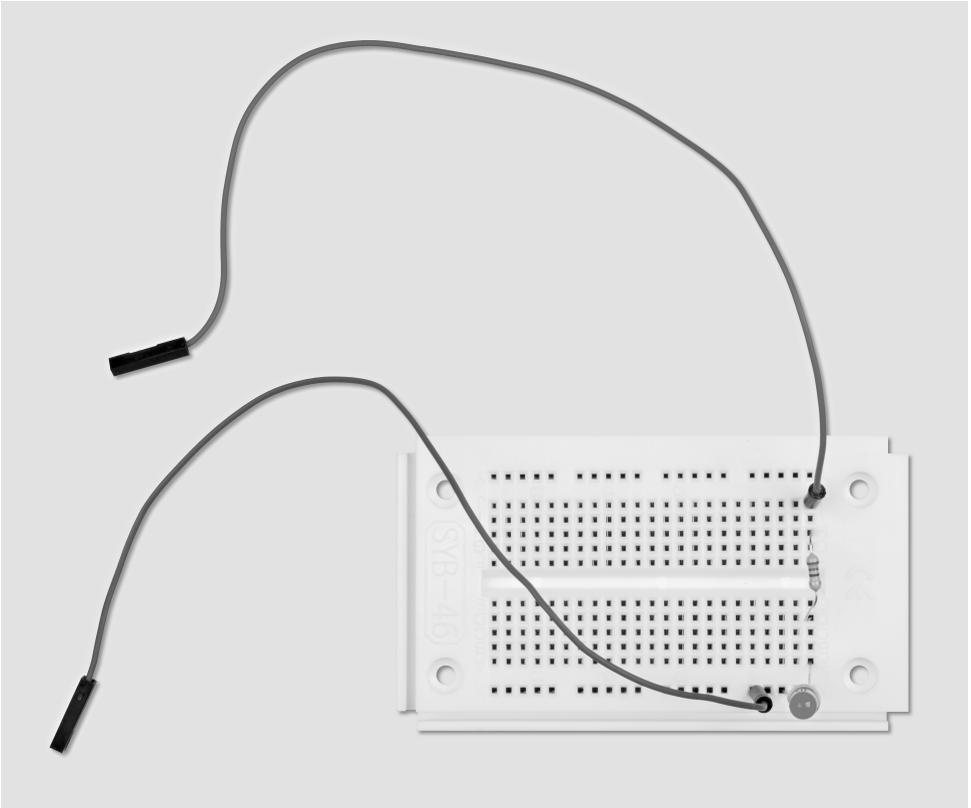


Fig. 6.2: Construction de la carte avec une LED

Composants nécessaires :
1x Carte de circuit imprimé
1x LED jaune
1x résistance de 220 Ω
2x Câbles de connexion

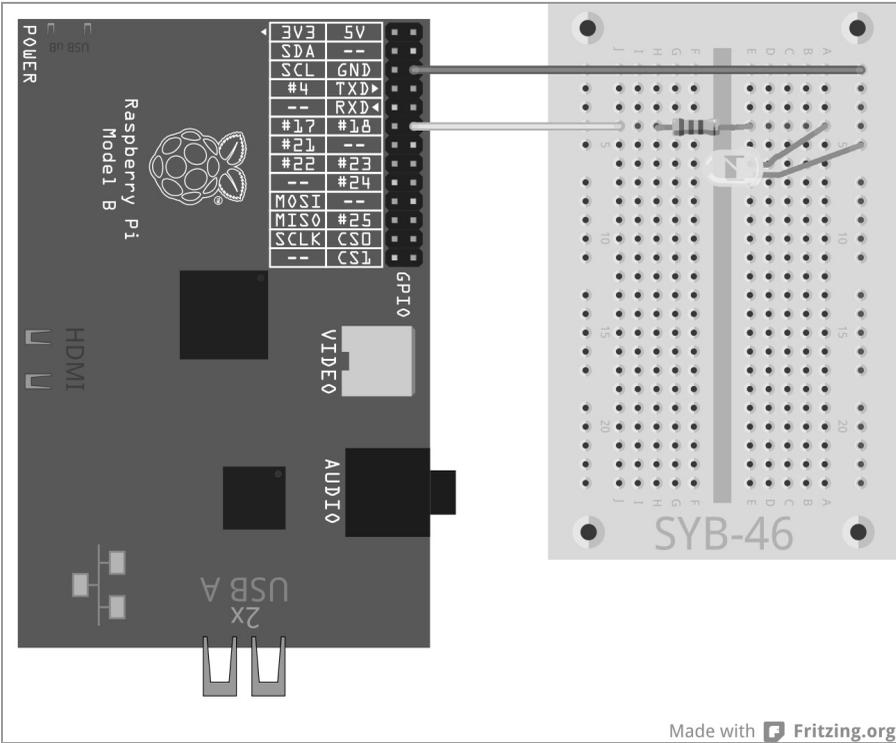


Fig. 6.3: Une LED sur le port GPIO 18

Le programme `leddimmen01.py` fait varier l'intensité lumineuse de la LED cycliquement et utilise pour ce faire une fonctionnalité propre PWM de la bibliothèque GPIO. Le signal PWM est généré comme un processus spécifique. De cette façon, une LED dont l'intensité lumineuse varie (presque) peut être utilisée comme une source lumineuse normale dans un programme.

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM); LED = 18
GPIO.setup(LED, GPIO.OUT)
print ("Ctrl+C arrête le programme")
p = GPIO.PWM(LED, 50); p.start(0)
try:
    while True:
        for c in range(0, 101, 2):
            p.ChangeDutyCycle(c); time.sleep(0.1)
        for c in range(100, -1, -2):
            p.ChangeDutyCycle(c); time.sleep(0.1)
except KeyboardInterrupt:
    p.stop(); GPIO.cleanup()
```

6.1.1 Voilà comment cela fonctionne

Une partie de ce programme vous est familière, certains éléments ne vous disent rien car nous faisons ici un petit tour dans la programmation orientée objet. Les bibliothèques sont importées au début, comme vous le savez déjà. Cette fois, une seule variable, `LED`, est fixée pour le port GPIO 18, qui est initialisé comme une sortie.

```
print("Ctrl+C arrête le programme")
```

Étant donné que ce programme avec une construction `try...except` s'exécute et doit être arrêté par l'utilisateur, des informations correspondantes s'affichent à l'écran pour l'utilisateur.

`p = GPIO.PWM(LED, 50)` La fonction `GPIO.PWM()` de la bibliothèque GPIO est essentielle pour la sortie des signaux PWM. Cette fonction nécessite deux paramètres, le port GPIO et la fréquence du signal PWM. Dans notre cas, le port GPIO est fixé avec la variable `LED` et la fréquence est de 50 hertz (Hz) (oscillations par seconde).

Pourquoi est-ce que la fréquence de 50 Hz est idéale pour PWM ?

L'œil humain ne perçoit pas bien les changements de lumière plus rapides que 20 Hz. Étant donné que le courant alternatif en Europe utilise une fréquence de 50 Hz, de nombreuses sources lumineuses clignotent à cette fréquence et ce clignotement n'est pas perceptible pour l'œil humain. Si une LED clignote à plus de 20 Hz mais moins de 50 Hz, des interférences avec d'autres sources lumineuses peuvent se produire et perturber l'effet perçu de variation d'intensité lumineuse.

`GPIO.PWM()` génère un soi-disant objet qui est enregistré dans la variable `p`. Ces objets sont davantage que de simples variables. Les objets peuvent avoir des propriétés différentes et sont influencés par des soi-disant méthodes. Les méthodes, séparées par un point, sont spécifiées directement après le nom de l'objet.

`p.start(0)` La méthode `start()` lance la génération du signal PWM. Pour ce faire, un rapport cyclique doit encore être spécifié. Dans notre cas, le rapport cyclique est 0 et la LED est toujours éteinte. La boucle infinie commence maintenant dans laquelle deux boucles sont imbriquées directement successivement, ce qui permet de faire varier l'intensité de la LED alternativement entre une intensité forte et une intensité faible.

`for c in range(0, 101, 2):` La boucle compte par incrément de 2 depuis 0 jusqu'à 100. La valeur est toujours spécifiée pour arrêter une boucle `for`, qui n'est pas atteinte en pratique, dans notre cas : 101.

`p.ChangeDutyCycle(c)` La méthode `ChangeDutyCycle()` définit à chaque cycle de boucle le rapport cyclique de l'objet PWM sur la valeur du compteur de boucle, également augmenté de 2 % à chaque fois, jusqu'à ce que dernier cycle soit à 100 % et que la LED éclaire avec sa pleine luminosité.

`time.sleep(0.1)` On attend 0,1 seconde à chaque cycle de boucle avant que le cycle suivant n'augmente le rapport de cycle de 2 %.

```
for c in range(100, -1, -2):
    p.ChangeDutyCycle(c); time.sleep(0.1)
```

Après que la LED a atteint sa pleine luminosité, une deuxième boucle régule son intensité dans le sens inverse selon le même schéma. Cette boucle compte à rebours à partir de 100, avec des décréments de -2. Ce cycle se répète jusqu'à ce que l'utilisateur appuie sur la combinaison de touches `Ctrl + C`.

```
except KeyboardInterrupt:  
    p.stop(); GPIO.cleanup()
```

Le `KeyboardInterrupt` déclenche immédiatement la méthode `stop()` de l'objet PWM. Cette méthode arrête la génération du signal PWM. Après cela, les ports GPIO sont réinitialisés, comme nous le savons déjà avec les programmes précédents.

6.1.2 Faire varier l'intensité lumineuse de deux LED indépendamment l'une de l'autre

Étant donné qu'aucun temps de programmation dans le script Python n'est nécessaire pour programmer le signal PWM, il est possible de faire varier l'intensité de plusieurs LED indépendamment les unes des autres, comme le montre l'expérience suivante. Pour ce faire, branchez une autre LED via une pré-résistance sur le port GPIO 25.

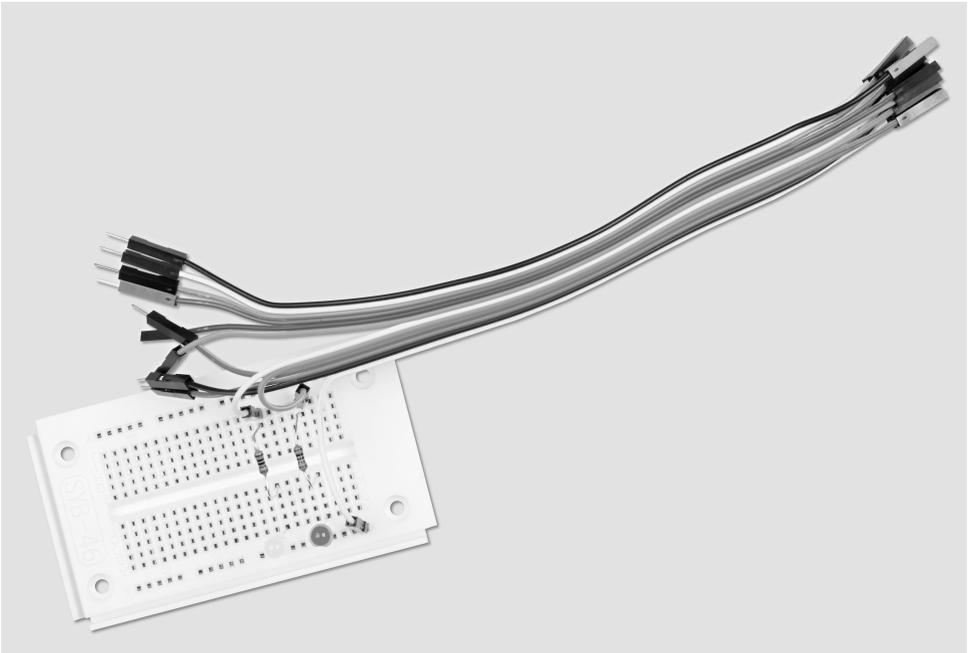


Fig. 6.4: Construction de la carte pour faire varier l'intensité lumineuse de deux LED.

- Composants nécessaires :
- 1x Carte de circuit imprimé
 - 1x LED jaune
 - 1x LED rouge
 - 2x Résistances de 220 Ω
 - 3x Câbles de connexion

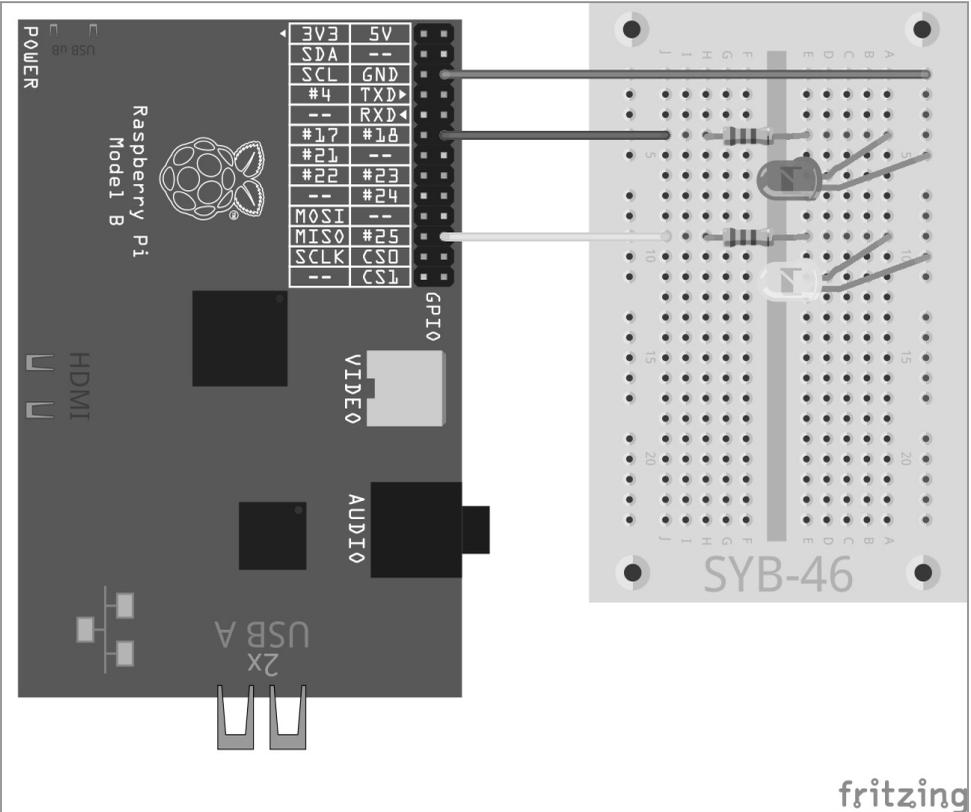


Fig. 6.5: Une deuxième LED sur le port GPIO 25

Le programme `leddimmen02.py` fait varier cycliquement l'intensité d'une LED pendant que l'autre LED éclaire avec une forte intensité avec la première LED, puis ne diminue pas d'intensité dans un autre cycle, éclaire de nouveau avec 0 et clignote rapidement.

```
import RPi.GPIO as GPIO
import time
```

```

GPIO.setmode(GPIO.BCM); LED = [18,25]
GPIO.setup(LED[0], GPIO.OUT); GPIO.setup(LED[1], GPIO.OUT)

print ("Ctrl+C arrête le programme")

p = GPIO.PWM(LED[0], 50); q = GPIO.PWM(LED[1], 50)
p.start(0)
q.start(0)

try:
    while True:
        for c in range(0, 101, 2):
            p.ChangeDutyCycle(c); q.ChangeDutyCycle(c)
            time.sleep(0.1)
        q.ChangeFrequency(10)
        for c in range(0, 101, 2):
            p.ChangeDutyCycle(100-c); q.ChangeDutyCycle(c)
            time.sleep(0.1)
        q.ChangeFrequency(50)
except KeyboardInterrupt:
    p.stop(); GPIO.cleanup()

```

6.1.3 Voilà comment cela fonctionne

La structure de base du programme correspond à celle de l'expérience précédente avec quelques petites améliorations.

```
LED = [18,25]; GPIO.setup(LED[0], GPIO.OUT); GPIO.setup(LED[1], GPIO.OUT)
```

A la place d'un variable pour le port GPIO, une liste est maintenant définie avec deux variables et les deux ports GPIO 18 et 25 sont initialisés comme des sorties pour les LED.

```
p = GPIO.PWM(LED[0], 50); q = GPIO.PWM(LED[1], 50); p.start(0); q.start(0)
```

Ensuite les deux objets `p` et `q` sont également créés pour générer les signaux PWM pour les deux LED avec une fréquence de 50 Hz pour chacune.

```

for c in range(0, 101, 2):
    p.ChangeDutyCycle(c); q.ChangeDutyCycle(c)
    time.sleep(0.1)

```

Dans la première boucle, le rapport cyclique des deux objets PWM est augmenté simultanément, incrément par incrément. Les deux LED se comportent de la même façon dans cette phase.

`q.ChangeFrequency(10)` A la fin de cette boucle, lorsque les deux LED ont atteint la pleine luminosité, la fréquence du signal PWM est réduite de 10 Hz par la méthode `ChangeFrequency()`. Cette fréquence est encore perçue comme un clignotement pour l'œil humain.

```

for c in range(0, 101, 2):
    p.ChangeDutyCycle(100-c); q.ChangeDutyCycle(c)
    time.sleep(0.1)

```

La deuxième boucle commence maintenant, par souci de simplicité, cette fois également avec un comptage croissant. Pour la première LED, à partir de l'objet PWM p , dont l'intensité doit être réduite pas à pas dans ce cycle, la valeur correspondante pour le rapport cyclique est calculée à chaque cycle. Pour la deuxième LED, à partir de l'objet PWM q , le rapport cyclique est simplement augmenté à nouveau. L'effet de clignotement est causé par le changement de fréquence.

`q.ChangeFrequency(50)` A la fin de cette deuxième boucle, la fréquence de cette LED revient à 50 Hz pour qu'elle soit dans le cycle suivant à nouveau aussi lumineux progressivement que la première LED.

7 Indicateur à LED montrant l'espace disponible des cartes mémoires

Les cartes mémoires sont toujours pleines trop rapidement comme les disques durs. C'est pourquoi on souhaite avoir un indicateur visuel du remplissage de la carte mémoire pour pouvoir savoir en un clin d'œil s'il y reste peu d'espace disponible. Vous pouvez réaliser cela très simplement avec trois LED sur le Raspberry Pi. Pour ce faire, des fonctions du système d'exploitation sont utilisées et elles sont appelées par Python.



Fig. 7.1: L'espace mémoire libre peut évidemment être affiché directement dans le gestionnaire de fichiers sur le Raspberry Pi.

Pour afficher l'espace mémoire disponible, nous utilisons les trois LED du feu de signalisation qui s'allument avec différentes combinaisons de couleurs.

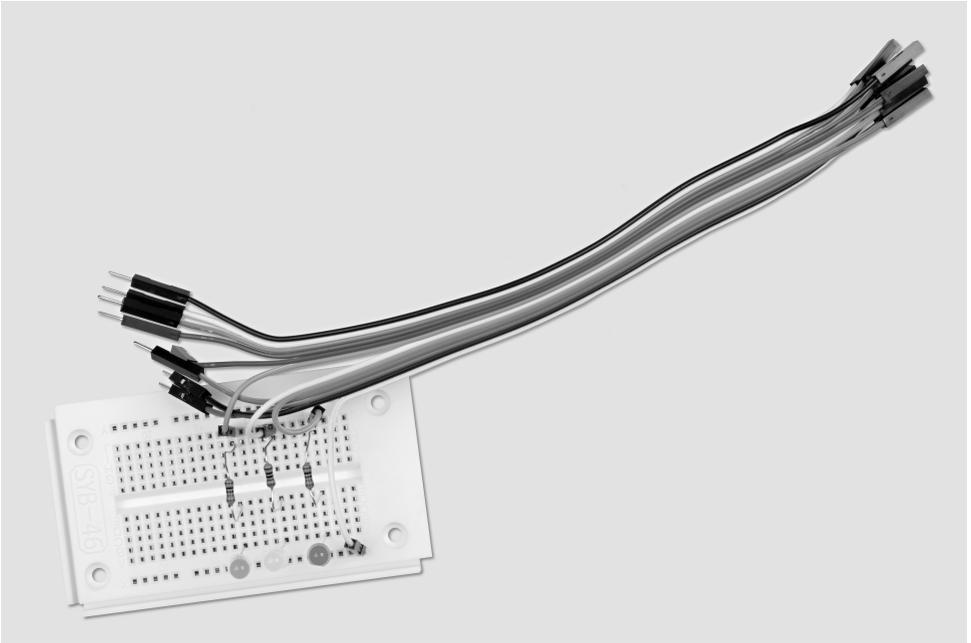
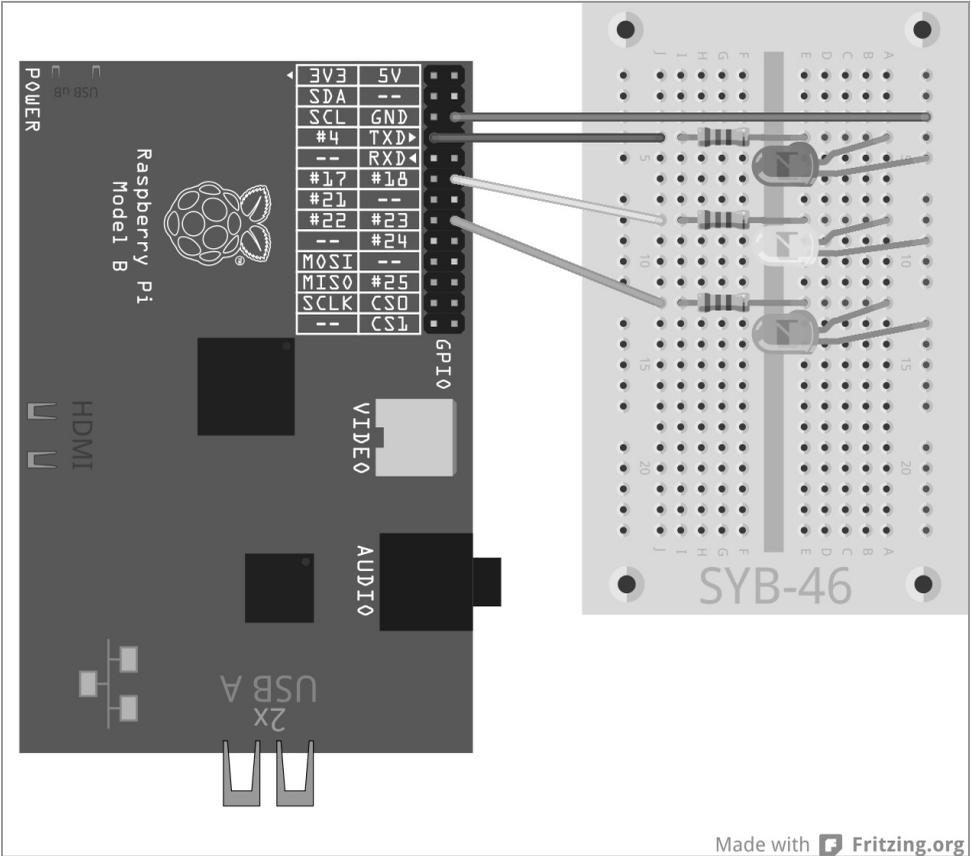


Fig. 7.2: Construction de la carte pour l'indicateur à LED montrant l'espace disponible dans les cartes mémoires.

Composants nécessaires :

- 1x Carte de circuit imprimé
- 1x LED rouge
- 1x LED jaune
- 1x LED verte
- 3x résistances de 220 Ω
- 4x Câbles de connexion



Made with Fritzing.org

Fig. 7.3: Les trois LED indiquent l'espace mémoire disponible sur la carte mémoire.

Le programme `speicheranzeige.py` fournit différents affichages par LED en fonction de l'espace mémoire libre sur la carte mémoire :

Espace mémoire libre	Affichage par LED
< 1 Mo	Rouge
De 1 Mo à 10 Mo	Rouge-jaune
De 10 Mo à 100 Mo	Jaune
De 100 Mo à 500 Mo	Jaune-Vert
> 500 Mo	Vert

Tableau 7.1: L'état de remplissage de la carte mémoire est ainsi affiché.

```

import RPi.GPIO as GPIO
import time
import os

g1 = 1; g2 = 10; g3 = 100; g4 = 500

GPIO.setmode(GPIO.BCM)
LED = [4,18,23]
for i in range(3):
    GPIO.setup(LED[i], GPIO.OUT, initial=False)

print ("Ctrl+C arrête le programme")

try:
    while True :
        s = os.statvfs('/')
        f = s.f_bsize * s.f_bavail / 1000000

        if f < g1:
            x = "100"
        elif f < g2:
            x = "110"
        elif f < g3:
            x = "010"
        elif f < g4:
            x = "011"
        else:
            x = "001"

        for i in range(3):
            GPIO.output(LED[i], int(x[i]))
            time.sleep(1.0)

except KeyboardInterrupt:
    GPIO.cleanup()

```

Si vous exécutez le programme, les LED indiquent en permanence l'espace mémoire libre qui reste disponible sur la carte mémoire. Testez-le en copiant et supprimant de gros fichiers sur la carte mémoire via le réseau. L'affichage se met à jour automatiquement.

7.1.1 Voilà comment cela fonctionne

Le programme utilise le module `os` de Python pour calculer l'espace mémoire libre qui est disponible à partir des fonctions du système d'exploitation.

`import os` Le module `os` doit être importé au début du programme, comme les autres modules.

`g1 = 1; g2 = 10; g3 = 100; g4 = 500` Ces lignes définissent les limites de la plage disponible pour l'espace mémoire, entre lesquelles l'affichage doit changer. Par souci de simplicité, le programme utilise des mégaoctets et non des octets car l'on imagine mieux ces chiffres. Vous pouvez à tout moment modifier les limites. Les quatre valeurs doivent seulement être placées dans un ordre croissant.

```
GPIO.setmode(GPIO.BCM)
LED = [4,18,23]
for i in range(3):
    GPIO.setup(LED[i], GPIO.OUT, initial=False)
```

Une liste définit le numéro de port GPIO des trois LED. Après cela, une boucle initialise les trois ports GPIO comme des sorties et toutes les LEDs sont éteintes.

Également dans cette expérience, nous utilisons une construction `try...except` et une boucle infinie pour que le programme s'exécute et se répète toujours automatiquement, jusqu'à ce que l'utilisateur l'arrête en appuyant sur `[Ctrl] + [C]`. Les fonctions vraiment intéressantes suivent ensuite. Elles accèdent au système d'exploitation et vérifient l'espace mémoire libre disponible.

`s = os.statvfs('/')` Le module statistique `os.statvfs()` à partir de la bibliothèque `os` fournit différentes informations statistiques sur le système de fichiers qui sont écrites ici dans la boucle infinie à chaque cycle de boucle à nouveau comme un objet dans la variables.

`f = s.f_bsize * s.f_bavail / 1048576` La méthode `s.f_bsize` fournit ici la taille d'un bloc mémoire en octet. `s.f_bavail` donne le nombre de blocs libres. Le produit à partir des deux valeurs donne donc le nombre d'octets libres, qui est divisé ici par 1 048 576, pour obtenir le nombre de mégaoctets libres. Le résultat est enregistré dans la variable `f`.

```
if f < g1:
    x = "100"
```

Si l'espace mémoire libre est inférieur à la première valeur limite (1 Mo), la chaîne de caractères `x`, qui indique le motif des LED allumées, est spécifiée égale à "100". La première LED rouge doit s'allumer. Le motif est une chaîne de caractères simple composée des chiffres 0 et 1.

```
elif f < g2:
    x = "110"
elif f < g3:
    x = "010"
elif f < g4:
    x = "011"
```

À l'aide de la requête `elif`, les autres limites sont vérifiées et le motif des LED est fixé en conséquence. Lorsque la première question n'a pas de réponse/objet, il y a également plus de 1 Mo d'espace libre disponible.

```
else:
    x = "001"
```

Si aucune requête n'est satisfaite, alors il y a encore plus d'espace libre que la limite supérieure spécifiée et le motif des LED est fixé à "001". La dernière LED verte doit s'allumer.

```
for i in range(3):
    GPIO.output(LED[i], int(x[i]))
```

Une boucle définit/fixe les valeurs de sortie pour les trois LED. Toutes les LED obtiennent successivement la valeur numérique de chaque chiffre à partir de la chaîne de caractères 0 ou 1. Les valeurs 0 et 1 peuvent également être utilisées comme `False` et `True` pour activer ou inactiver les sorties GPIO. La fonction `int()` calcule à partir d'un chiffre de ces valeurs numériques. Le chiffre est lu par le compteur de boucle `i` à partir d'une certaine position de la chaîne de caractères du motif.

`time.sleep(1.0)` Le programme attend 1 seconde jusqu'au cycle suivant de boucle. Pour économiser les performances, vous pouvez également spécifier des temps d'attente plus long jusqu'à ce que le calcul de l'espace mémoire libre doive se répéter.

À cet endroit, la boucle `while...True` recommence depuis le début. Si l'utilisateur doit appuyer entre temps sur la combinaison de touches `[Ctrl] + [C]`, un `KeyboardInterrupt` est déclenché et la boucle est quittée. Après cela, les ports GPIO sont fermés et les LED éteintes.

8 Dé graphique

Un jeu sympathique offre des graphismes et pas seulement une réponse textuelle comme à l'époque des premiers ordinateurs DOS. La bibliothèque PyGame fournit des fonctions et des objets prédéfinis pour l'affichage graphique et la programmation de jeux. Ainsi, vous n'avez pas à tout créer à partir de zéro.

On a besoin d'un dé pour jouer à de nombreux jeux mais il est fréquent de ne pas en avoir sous la main. L'exemple suivant de programme montre combien il est facile d'utiliser le Raspberry Pi comme un dé à l'aide de Python et PyGame.

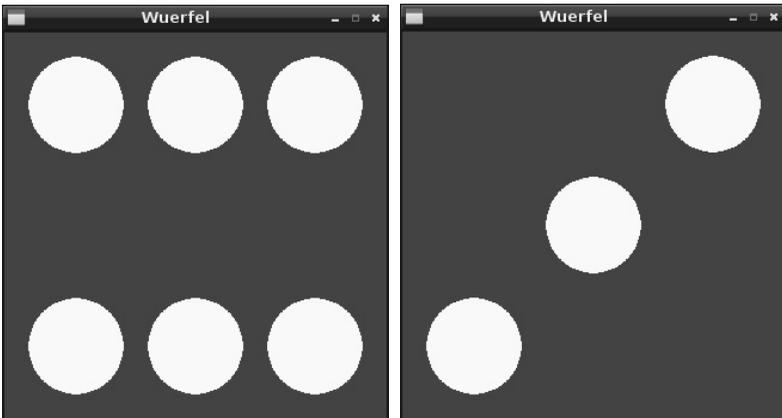


Fig. 8.1: Utiliser le Raspberry Pi comme un dé

Le dé doit être aussi simple que possible et être utilisé avec une seule touche, et le résultat aléatoire du lancer de dé doit être affiché graphiquement comme un « vrai » dé. Le programme suivant `wuerfel.py` simule un tel dé sur l'écran.

```

# -*- coding: utf-8 -*-
import pygame, sys, random
from pygame.locals import *
pygame.init()

FELD = pygame.display.set_mode((320, 320))
pygame.display.set_caption("Dé")

BLEU = (0, 0, 255); BLANC = (255, 255, 255)
P1 = ((160, 160)); P2 = ((60, 60)); P3 = ((160, 60));
P4 = ((260, 60))
P5 = ((60, 260)); P6 = ((160, 260)); P7 = ((260, 260))
mainloop = True

print "Appuyer sur n'importe quelle touche pour lancer le dé, sur [échap]
pour quitter le jeu"

while mainloop:
    for event in pygame.event.get():
        if event.type == QUIT or (event.type == KEYUP
and event.key == K_ESCAPE):
            mainloop = False
        if event.type == KEYDOWN:
            CHAMP.fill(BLEU)
            CHIFFRE = random.randrange(1, 7); print CHIFFRE
            if CHIFFRE == 1:
                pygame.draw.circle(CHAMP, BLANC, P1, 40)
            if CHIFFRE == 2:
                pygame.draw.circle(CHAMP, BLANC, P2, 40)
                pygame.draw.circle(CHAMP, BLANC, P7, 40)
            if CHIFFRE == 3:
                pygame.draw.circle(CHAMP, BLANC, P1, 40)
                pygame.draw.circle(CHAMP, BLANC, P4, 40)
                pygame.draw.circle(CHAMP, BLANC, P5, 40)
            if CHIFFRE == 4:
                pygame.draw.circle(CHAMP, BLANC, P2, 40)
                pygame.draw.circle(CHAMP, BLANC, P4, 40)
                pygame.draw.circle(CHAMP, BLANC, P5, 40)
                pygame.draw.circle(CHAMP, BLANC, P7, 40)
            if CHIFFRE == 5:
                pygame.draw.circle(CHAMP, BLANC, P1, 40)
                pygame.draw.circle(CHAMP, BLANC, P2, 40)
                pygame.draw.circle(CHAMP, BLANC, P4, 40)
                pygame.draw.circle(CHAMP, BLANC, P5, 40)
                pygame.draw.circle(CHAMP, BLANC, P7, 40)
            if CHIFFRE == 6:
                pygame.draw.circle(CHAMP, BLANC, P2, 40)
                pygame.draw.circle(CHAMP, BLANC, P3, 40)
                pygame.draw.circle(CHAMP, BLANC, P4, 40)
                pygame.draw.circle(CHAMP, BLANC, P5, 40)
                pygame.draw.circle(CHAMP, BLANC, P6, 40)
                pygame.draw.circle(CHAMP, BLANC, P7, 40)
    pygame.display.update()
pygame.quit()

```

Fonctionne sans sudo

Étant donné que ce programme n'a besoin d'aucun port GPIO, il fonctionne aussi sans privilège de super-utilisateur. Vous pouvez facilement lancer l'IDE de Python via le symbole de bureau *IDLE*.

8.1.1 Voilà comment cela fonctionne

Ce programme affiche de nombreuses nouvelles fonctions, en particulier pour les affichages graphiques avec la bibliothèque PyGame qui peut être évidemment utilisée pour les jeux mais également pour afficher d'autres graphismes à l'écran.

```
import pygame, sys, random
from pygame.locals import *
pygame.init()
```

Ces trois lignes de programme sont placées au début de presque tous les programmes qui utilisent PyGame. En plus du module `random` que vous connaissez déjà pour générer un chiffre aléatoire, le module `pygame` et le module `sys` sont chargés, car ils contiennent des fonctions système importants de PyGame, comme p. ex. l'ouverture et la fermeture de fenêtres. Toutes les fonctions de la bibliothèque PyGame sont importées puis le vrai module PyGame est initialisé.

```
CHAMP = pygame.display.set_mode((320, 320))
```

Cette fonction importante dans chaque programme qui utilise un affichage graphique, définit une zone de dessin, une dite surface, qui fait la taille de 320 x 320 pixels dans notre exemple et qui reçoit le nom `CHAMP`. Notez que l'écriture entre les doubles crochets est utilisée essentiellement pour les coordonnées graphiques de l'écran. Cette surface est affichée dans une nouvelle fenêtre à l'écran.

```
pygame.display.set_caption("Dé")
```

Cette ligne présente le nom de la fenêtre.

```
BLEU = (0, 0, 255); BLANC = (255, 255, 255)
```

Ces lignes définissent les deux couleurs utilisées, le bleu et le blanc. Vous pouvez également spécifier à chaque fois dans le programme les valeurs de couleurs, ce qui ne contribue cependant pas directement à l'aperçu.

Affichage des couleurs à l'écran

Les couleurs sont définies dans Python comme dans la plupart des autres langages de programmation, avec trois chiffres compris entre 0 et 255, qui spécifient chacun la composante rouge, verte et bleu de la couleur. Les écrans utilisent un mélange de couleurs qui s'additionnent, et les trois composantes de couleur donnent ensemble du blanc à pleine saturation.

```
P1 = ((160, 160)); P2 = ((60, 60)); P3 = ((160, 60)); P4 = ((260, 60)); P5 = ((60, 260)); P6 = ((160, 260)); P7 = ((260, 260))
```

Ces lignes définissent le centre du point du dé. Les trois axes du point du dé sont placés sur la zone de dessin de 320 x 320 pixels avec les coordonnées 60, 160 et 260.

Le système de coordonnées pour l'infographie

Chaque point dans une fenêtre ou sur un objet-surface est représenté par ses coordonnées x et y. L'origine du système de coordonnées n'est pas à gauche en bas comme on l'apprend à l'école mais à gauche en haut. Exactement comme on lit un texte de gauche à droite et du haut vers le bas, l'axe des x va de gauche à droite et l'axe des y de haut en bas.

Les sept points P1 à P7 désignent les centres du dé indiqués dans le graphique. Chaque point du dé a un rayon de 40 pixels. Avec un écart axial de 80 pixels, il reste 20 pixels entre les points du dé et 20 pixels jusqu'au bord de la fenêtre.

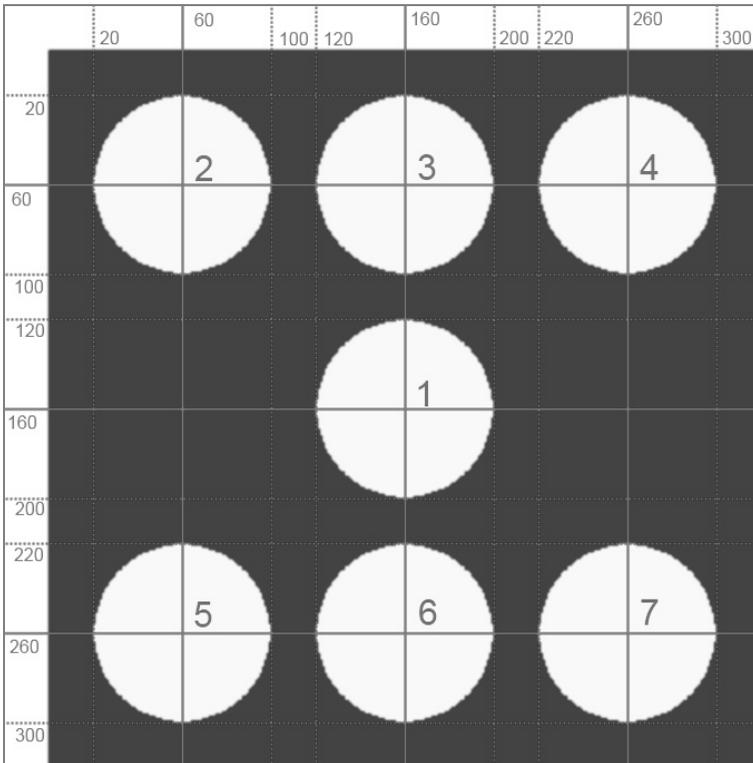


Fig. 8.2: Les points du dé et leurs coordonnées

À cet endroit, une variable nommée `mainloop` est encore spécifiée sur `True` avec les autres variables. Cette variable sera nécessaire plus tard pour la boucle principale du jeu.

`mainloop = True` Les bases sont ainsi créées et le vrai jeu peut commencer.

```
print "Appuyer sur n'importe quelle touche pour lancer le dé, sur [échap]  
pour quitter le jeu"
```

Cette ligne explique brièvement à l'utilisateur ce qu'il faut faire. Chaque fois que vous appuyez sur une touche du clavier, vous relancez le dé. `print` écrit toujours dans la fenêtre Python-Shell, et non dans la nouvelle fenêtre graphique.

`while mainloop`: La boucle principale du jeu commence maintenant. Dans de nombreux jeux, une boucle infinie est utilisée. Elle est toujours répétée et vérifie en permanence l'activité de l'utilisateur. Une condition d'arrêt doit être définie quelque part dans la boucle pour s'assurer que le jeu puisse être arrêté.

La variable `mainloop` est utilisée ici et elle prend en compte uniquement les deux valeurs booléennes `True` et `False` (vrai et faux, marche et arrêt). Elle est spécifiée comme `True` au début et est vérifiée à chaque cycle de boucle. Si, pendant une boucle, elle prend la valeur `False`, la boucle est quittée avant le cycle suivant.

`for event in pygame.event.get()`: Cette ligne lit la dernière activité de l'utilisateur et s'enregistre sous `event`. Dans le jeu, il y a uniquement deux types d'activité de l'utilisateur qui comptent pour le jeu : l'utilisateur appuie sur une touche et lance ainsi le dé, ou l'utilisateur souhaite quitter le jeu.

```
if event.type == QUIT or (event.type == KEYUP  
and event.key == K_ESCAPE):  
    mainloop = False
```

Il existe deux façons de quitter le jeu : on peut cliquer sur le signe x en haut à droite de la fenêtre ou appuyer sur le bouton `[échap]`. Si l'on clique sur le signe x, `event.type == QUIT` fourni par le système d'exploitation est exécuté. Si l'on appuie sur un bouton puis qu'on le relâche, `event.type == KEYUP` est exécuté. En outre, le bouton enfoncé est enregistré dans `event.key` dans ce cas.

La requête `if` déjà décrite vérifie si l'utilisateur veut fermer la fenêtre ou (or) s'il a appuyé sur un bouton puis l'a relâché et (and) si ce bouton est le bouton avec la désignation interne `K_ESCAPE`. Si c'est le cas, la variable `mainloop` est spécifiée sur `False`, ce qui arrête la boucle principale du jeu avant le cycle suivant.

`if event.type == KEYDOWN`: Le deuxième type d'activité de l'utilisateur qui se produit toujours plusieurs fois pendant le jeu (et jamais une seule fois), est le fait que l'utilisateur appuie sur une touche. Cela peut être n'importe quelle touche à l'exception de la touche `[échap]`. Dès que vous appuyez sur une touche (`KEYDOWN`), une partie importante du programme est lancée pour générer et afficher le résultat du lancer de dé.

`CHAMP.fill(BLEU)` Tout d'abord, l'objet-surface appelé `CHAMP`, la fenêtre de programme, prend la couleur définie comme `BLEU` au début, pour effacer le résultat du lancer de dé précédent.

```
CHIFFRE = random.randrange(1, 7)
```

La fonction aléatoire `random` génère maintenant un chiffre aléatoire compris entre 1 et 6 et l'enregistre dans la variable `CHIFFRE`.

`print CHIFFRE` Cette ligne écrit seulement pour contrôler le résultat du lancer de dé dans la fenêtre du Python-Shell. Vous pouvez également omettre cette ligne si vous souhaitez renoncer à l'affichage textuel.

```
if CHIFFRE == 1:
    pygame.draw.circle(CHAMP, BLANC, P1, 40)
```

Les six requêtes suivent maintenant le même schéma. Lorsque le chiffre aléatoire du lancer de dé a une certaine valeur, un à six points de dé sont dessinés en conséquence. La fonction `pygame.draw.circle()` utilisée à cette fin a besoin de quatre ou cinq paramètres :

- *Surface* donne la surface de dessin sur laquelle est dessiné par exemple la `CHAMP`.
- *Couleur* donne la couleur du cercle, par exemple la couleur `BLANC` précédemment définie.
- *Centre* donne le centre du cercle.
- *Rayon* donne le rayon du cercle.
- *Épaisseur* donne l'épaisseur de la ligne du cercle. Si ces paramètres sont omis ou spécifiés égaux à 0, le cercle est plein.

Si une des conditions `if` est satisfaite, les points de dé sont d'abord enregistrés uniquement sur une surface de dessin virtuelle.

`pygame.display.update()` Cette ligne à la fin de la boucle met à jour le graphique à l'écran. Les points du dé sont maintenant vraiment visibles.

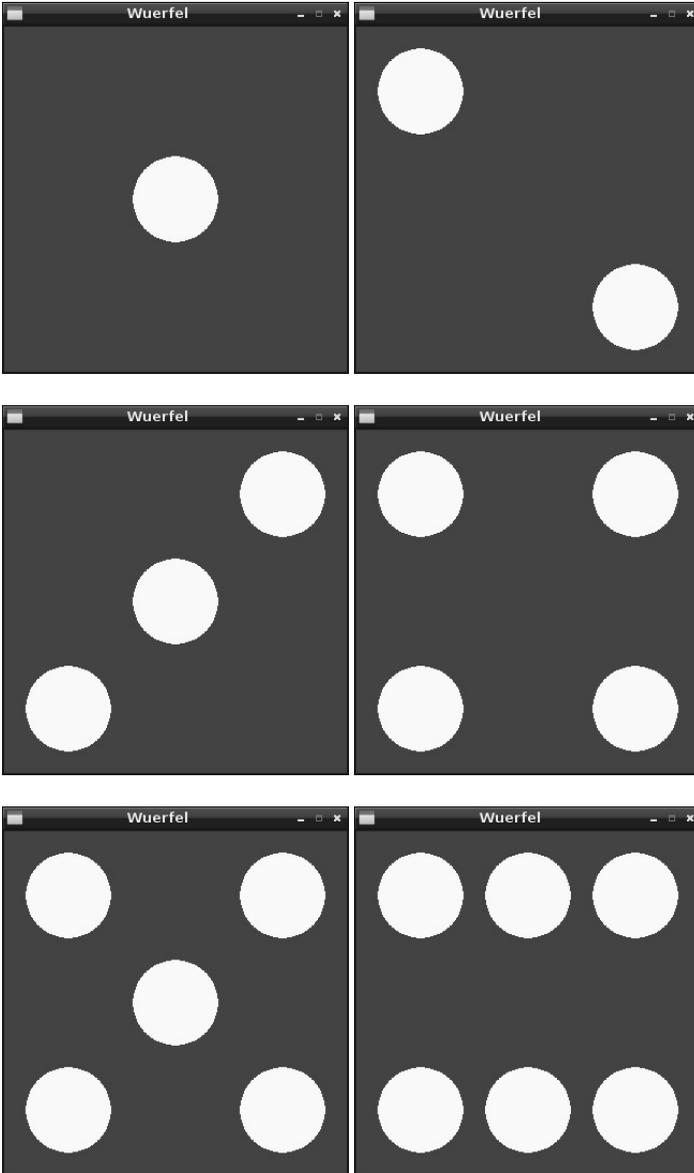


Fig. 8.3: Les six résultats possibles du lancer de dé.

La boucle recommence immédiatement et attend que l'utilisateur appuie sur une touche. Si pendant la boucle `mainloop` est spécifiée sur `False`, parce que l'utilisateur a arrêté le jeu, la boucle ne continue plus et la ligne suivante est exécutée à la place :

`pygame.quit()` Cette ligne arrête le module PyGame, ce qui ferme également la fenêtre graphique et par conséquent l'ensemble du programme.

9 Horloge analogique sur l'écran

L'affichage numérique de l'heure auquel nous nous sommes habitués aujourd'hui avec les ordinateurs, n'est devenu à la mode que dans les années 1970. On a lu l'heure pendant des siècles sur des horloges analogiques avec des aiguilles tournant dans un cadran. L'engouement pour l'horloge numérique a quelque peu diminué ces dernières années car on a reconnu que les horloges analogiques étaient lues plus rapidement et clairement en cas de mauvaises conditions météorologiques ou lorsqu'elles sont loin comme par exemple dans les gares. L'œil humain saisit un graphique plus rapidement que des chiffres ou des lettres. L'image d'une horloge analogique marque la mémoire à court terme de sorte que l'on peut l'interpréter correctement même lorsque l'on ne l'a vu de manière incomplète ou floue. Si l'on regarde une horloge numérique de manière imprécise, on ne peut pas tirer de conclusions fiables à propos de l'heure affichée.

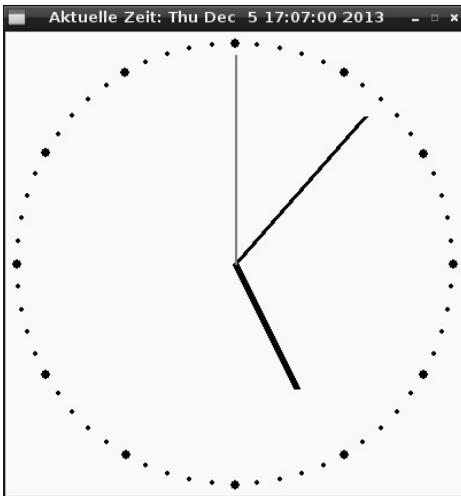


Fig. 9.1: Horloge analogique programmée avec PyGame.

Ce programme doit montrer non seulement comment on programme une horloge mais également expliquer les principes de base pour les affichages analogiques, comme ils peuvent être utilisés non seulement pour lire l'heure mais également pour afficher différentes valeurs mesurées ou données statistiques.

Trois aiguilles tournent au centre du cadran rond pour indiquer les heures, les minutes et les secondes. En haut, dans le titre de la fenêtre, vous trouverez en outre une horloge numérique qui affiche simultanément l'heure.

Le programme `uhr01.py` affiche une horloge analogique à l'écran :

```
import pygame, time
from pygame.locals import *
from math import sin, cos, radians
pygame.init()
ROUGE = (255, 0, 0); BLANC = (255, 255, 255); NOIR = (0, 0, 0)
CHAMP = pygame.display.set_mode((400, 400))
CHAMP.fill(BLANC)
MX = 200; MY = 200; MP = ((MX, MY))
def point(A, W):
    w1 = radians(W * 6 - 90); x1 = int(MX + A * cos(w1))
    y1 = int(MY + A * sin(w1)); return((x1, y1))
for i in range(60):
    pygame.draw.circle(CHAMP, NOIR, point(190, i), 2)
for i in range(12):
    pygame.draw.circle(CHAMP, NOIR, point(190, i * 5), 4)
mainloop = True; s1 = 0
while mainloop:
    temps = time.localtime()
    s = temps.tm_sec; m = temps.tm_min; h = temps.tm_hour
    if h > 12:
        h = h - 12
    hm = (h + m / 60.0) * 5
    if s1 <> s:
        pygame.draw.circle(CHAMP, BLANC, MP, 182)
        pygame.draw.line(CHAMP, NOIR, MP, point(120, hm), 6)
        pygame.draw.line(CHAMP, NOIR, MP, point(170, m), 4)
        pygame.draw.line(CHAMP, ROUGE, MP, point(180, s), 2)
        s1 = s
        pygame.display.set_caption("Heure actuelle : " +
time.asctime())
        pygame.display.update()
        for event in pygame.event.get():
            if event.type == QUIT or (event.type ==
KEYUP and event.key == K_ESCAPE):
                mainloop = False
pygame.quit()
```

9.1.1 Voilà comment cela fonctionne

Ce programme montre d'autres fonctions de la bibliothèque PyGame et la bibliothèque `time` ainsi que des fonctions trigonométriques simples, qui sont utilisées pour les affichages analogiques.

```
import pygame, time
from pygame.locals import *
from math import sin, cos, radians
pygame.init()
```

Au début, la bibliothèque PyGame est initialisée comme dans le dernier programme. En outre, la bibliothèque `time` et trois fonctions de la très vaste bibliothèque `math` sont importées pour la détermination du temps.

```
ROUGE = (255, 0, 0); BLANC = (255, 255, 255); NOIR = (0, 0, 0)
```

Les trois couleurs utilisées dans le graphisme sont enregistrées dans trois variables.

```
CHAMP = pygame.display.set_mode((400, 400)); CHAMP.fill(BLANC)
```

Une fenêtre de 400 x 400 pixel est ouverte et complètement colorée en blanc.

```
MX = 200; MY = 200; MP = ((MX, MY))
```

Trois variables déterminent les coordonnées du point central(centre)à partir duquel tous les autres éléments graphiques, le cadran et les aiguilles sont placés. Les variables `MX` et `MY` contiennent les coordonnées x et y du point central, la variable `MP` le point central comme point, tel qu'il est utilisé pour les fonctions graphiques.

La définition d'une fonction vient ensuite. Elle calcule la distance entre le centre et un des points d'angle dans le système de coordonnées. Cette fonction est appelée plusieurs fois dans le programme pour représenter à la fois le cadran et les aiguilles.

```
def point(A, W):  
    w1 = radians(W * 6 - 90); x1 = int(MX + A * cos(w1))  
    y1 = int(MY + A * sin(w1)); return((x1, y1))
```

La fonction utilise deux paramètres : A est la distance entre le point souhaité et le centre W est l'angle formé par rapport au centre. Pour simplifier la représentation dans le cas de l'horloge, nous prenons l'angle formé dans le sens antihoraire par rapport à l'axe vertical de 12 heures. L'angle n'est pas exprimé en degré mais en minutes, $1/60$ d'un tour complet, dans la fonction. Ces hypothèses permettent de ne pas faire de nombreux calculs intermédiaires.

Comme la plupart des langages de programmation, Python calcule les unités angulaires en radian et non en degré. La fonction `radian()` de la bibliothèque `math` convertit les degrés en radians. Pour ce faire, l'angle utilisé dans le programme en minute est multiplié par 6 pour obtenir des degrés puis on soustrait 90 degrés pour que l'axe vertical 0 pointe vers le haut, comme la minute 0 de chaque heure. Cet angle converti en radians est enregistré à l'intérieur de fonction dans la variable `w1` pour les calculs ultérieurs.

L'affichage d'une horloge analogique repose sur les fonctions trigonométriques sinus et cosinus. Elles déterminent les coordonnées à partir de l'angle d'un point en radian par rapport au point central (`w1`) dans le système de coordonnées rectangulaire (`x1` et `y1`). Les coordonnées du point central sont prises à partir des variables `MX` et `MY` qui sont définies en dehors de la fonction et appliquées globalement. La distance du point par rapport au point central est obtenue via le paramètre A de la fonction. La fonction `int()` est dérivée du résultat de la valeur entière, car les coordonnées de pixel peuvent être données seulement avec des entiers.

La valeur de retour de la fonction est un point géométrique avec les coordonnées calculées `x1` et `y1`, qui sont définies comme tous les points entre des parenthèses.

Le cadran est dessiné selon la définition de cette fonction.

```
for i in range(60):  
    pygame.draw.circle(CHAMP, NOIR, point(190, i), 2)
```

Une boucle dessine successivement les 60 points-minutes sur un cercle. Tous les points sont déterminés avec la fonction `point()`. Ils sont à la même distance du point central, qui est toujours éloigné de 10 pixels du bord de la fenêtre avec 190 pixels en quatre quadrants. Les points ont un rayon de 2 pixels.

```
for i in range(12):
    pygame.draw.circle(CHAMP, NOIR, point(190, i * 5), 4)
```

Une deuxième boucle dessine 12 cercles plus grands qui marquent les heures sur le cadran. Ils ont un rayon de 4 pixels, sont simplement dessinés sur le cercle précédent et les recouvrent complètement. Le symbole des heures se suivent à une distance angulaire de cinq unités-minutes que l'on peut obtenir en multipliant par 5 les données angulaires, qui sont données dans la fonction.

`mainloop = True`; `s1 = 0` Avant que la boucle principale du programme ne commence, deux variables auxiliaires sont encore définies. Elles sont nécessaires dans les processus suivants. `mainloop` indique comme dans le dernier exemple de programme si la boucle doit se répéter ou si l'utilisateur souhaite quitter le programme. `s1` enregistre la dernière seconde affichée.

```
while mainloop:
    temps = time.localtime()
```

La boucle principale du programme commence maintenant. Elle écrit à chaque cycle, indépendamment de sa durée, l'heure actuelle dans l'objet `temps`. Pour ce faire, la fonction `time.localtime()` de la bibliothèque `time` est utilisée. Le résultat est une structure de données qui est composée de différentes valeurs individuelles.

```
s = temps.tm_sec; m = temps.tm_min; h = temps.tm_hour
```

Les trois valeurs importantes pour les aiguilles, secondes, minutes et heures, sont écrites à partir de la structure dans trois variables `s`, `m` et `h`.

```
if h > 12:
    h = h - 12
```

L'horloge analogique ne montre que douze heures. La fonction `time.localtime()` fournit toutes les données de temps en format 24 heures. Pour afficher l'heure de l'après-midi, il suffit de soustraire 12 heures.

Affichage de l'heure avec une horloge analogique

Selon le mécanisme utilisé, il existe deux affichages différents pour les horloges analogiques. Dans une vraie montre analogique, l'aiguille des minutes effectue un mouvement circulaire uniforme. Dans une montre à commande numérique, comme par exemple les horloges de gare, une minute complète passe à la minute suivante. Cette dernière méthode a l'avantage de pouvoir lire l'heure en un clin d'œil, à la minute près. Les fractions de minutes ne sont pas importantes normalement pour la vie de tous les jours. Nous utilisons également cette méthode pour notre programme d'horloge. L'aiguille des heures doit cependant effectuer un mouvement circulaire uniforme. Il serait très étrange et déroutant si chaque heure écoulée pleinement sautait directement à l'heure suivante.

$hm = (h + m / 60.0) * 5$ La variable `hm` enregistre l'angle de l'aiguille des heures en unité-minute telle qu'elle est utilisée dans l'ensemble du programme. Pour ce faire, 1/60 de la valeur des minutes est ajoutée à l'heure actuelle. À chaque minute, l'aiguille de l'heure se déplace 1/60 d'une heure. La valeur calculée est multipliée par 5 parce que l'aiguille des heures avance de 5 unités-minutes sur le cadran en une heure.

`if s1 <> s` : La durée d'un cycle de boucle n'est pas connue dans le programme. Pour l'horloge analogique, cela signifie que le graphisme ne doit pas être mis à jour à chaque cycle de boucle mais uniquement lorsque la seconde actuelle est différente de la dernière affichée. Pour ce faire, la seconde affichée est enregistrée plus tard dans le programme dans la variable `s1`, la seconde actuelle reste toujours dans la variable `s`.

Si la seconde a changé par rapport à la dernière seconde affichée, les graphismes de l'horloge sont mis à jour avec les instructions qui suivent. Si elle n'a pas changé, le graphisme n'a pas besoin d'être mis à jour et la boucle recommence avec une autre requête de l'heure système actuelle.

```
pygame.draw.circle(CHAMP, BLANC, MP, 182)
```

Une surface circulaire blanche est d'abord dessinée en couvrant complètement les aiguilles. Le rayon de 182 pixels est légèrement plus grand que l'aiguille la plus longue de sorte qu'il ne reste plus rien. Il est beaucoup plus facile de dessiner un cercle complet, que de recouvrir au pixel près la dernière aiguille dessinée.

```
pygame.draw.line(CHAMP, NOIR, MP, point(120, hm), 6)
```

La ligne dessine l'aiguille des heures comme une ligne d'une largeur de 6 pixels et d'une longueur de 120 pixels à partir du point central avec un angle qui est spécifié par la variable `hm`. La fonction `pygame.draw.line()` n'a pas été utilisée jusqu'à présent. Elle a besoin de cinq paramètres :

- *Surface* donne la surface de dessin sur laquelle est dessiné par exemple le `CHAMP`.
- *Couleur* donne la couleur du cercle, par exemple la couleur `NOIR` précédemment définie.
- *Point de départ* indique le point de départ de la ligne ; dans cet exemple, il s'agit du centre de l'horloge.
- *Point final* indique le point final de la ligne ; dans l'exemple, il est calculé avec la fonction `point()` à partir de l'angle de l'aiguille des heures.
- *Épaisseur* donne l'épaisseur de la ligne.

La même fonction dessine également les deux autres aiguilles de l'horloge.

```
pygame.draw.line(CHAMP, NOIR, MP, point(170, m), 4)
```

Cette ligne dessine l'aiguille des minutes avec une ligne large de 4 pixels et longue de 170 pixels à partir du point central, avec un angle qui est donné par la valeur des minutes.

```
pygame.draw.line(CHAMP, ROUGE, MP, point(180, s), 2)
```

Cette ligne dessine l'aiguille des secondes comme une ligne rouge large de 2 pixels et longue de 180 pixels à partir du point central, avec un angle qui est donné par la valeur des secondes.

`s1 = s` La seconde qui vient d'être affichée est maintenant dans la variable `s1`, pour comparer cette valeur et la seconde actuelle dans le prochain cycle de boucle.

```
pygame.display.set_caption("Heure actuelle : " +
time.asctime())
```

Cette ligne écrit l'heure actuelle sous forme numérique dans le titre de la fenêtre. Pour ce faire, la fonction `time.asctime()` de la bibliothèque `time` est utilisée. Elle donne l'heure comme une chaîne de caractères entièrement formatée.

`pygame.display.update()` Les éléments graphiques sont dessinés seulement virtuellement jusqu'à présent. En premier lieu, cette ligne reconstruit vraiment l'affichage graphique. La mise à jour s'effectue en même temps. Par conséquent, on ne voit aucun vacillement lorsque les aiguilles sont dessinées les unes après autres.

```
for event in pygame.event.get():
    if event.type == QUIT or (event.type ==
KEYUP and event.key == K_ESCAPE):
        mainloop = False
```

Toujours dans la requête `if` et également seulement une fois dans la seconde, la requête relativement gourmande en ressources exécute les événements du système pour déterminer ici si l'utilisateur veut fermer la fenêtre de l'horloge dans la dernière seconde ou s'il a appuyé sur la touche `Échap`. Si c'est le cas, la variable `mainloop` est spécifiée sur `False` et la boucle n'est plus répétée.

`pygame.quit()` Cette dernière ligne arrête d'abord le module PyGame, ce qui ferme également la fenêtre graphique et par conséquent, l'ensemble du programme.

10 Boîte de dialogue graphique pour contrôler le programme

Aucun programme moderne qui requiert une interaction avec l'utilisateur, ne fonctionne en mode textuel pur. Il y a partout des interfaces graphiques sur lesquelles l'on peut cliquer sur des boutons au lieu d'avoir à entrer des données via le clavier.

Python n'offre pas lui-même d'interface graphique pour le programme. Il existe cependant plusieurs modules externes semblables à PyGame déjà décrit, qui sont spécialement conçus pour créer des interfaces graphiques. Un des modules les plus connus est *Tkinter*, qui offre à Python l'interface graphique *Tk*, qui peut également être utilisé pour divers autres langages de programmation.

Les structures de la boîte à outils graphiques Tk diffèrent légèrement de Python et peuvent sembler inhabituelles de prime abord. Par conséquent, nous commençons par un exemple très simple : Une LED doit être allumée et éteinte dans une boîte de dialogue.

- Composants nécessaires :
- 1x Carte de circuit imprimé
 - 1x LED rouge
 - 1x Résistance de 220 Ω
 - 2x Câbles de connexion

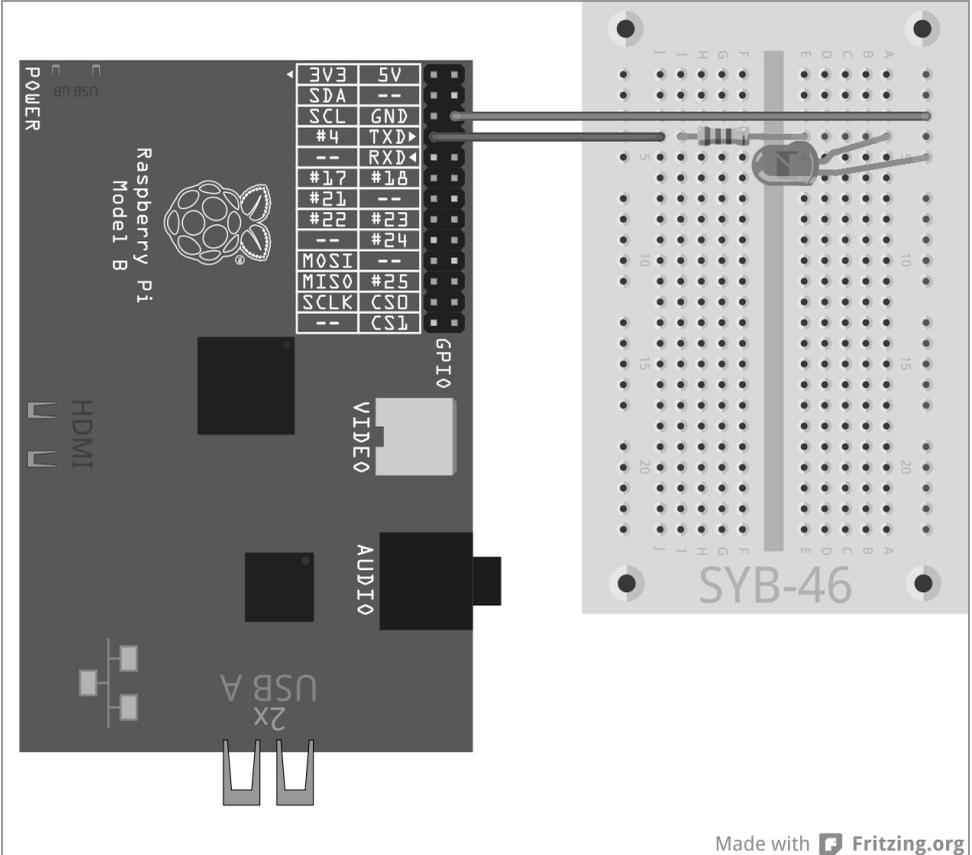


Fig. 10.1: Une LED unique sur le port GPIO 4

Branchez une LED via une pré-résistance sur le port GPIO 4. Le programme `ledtk01.py` va allumer cette LED.

```
import RPi.GPIO as GPIO
from Tkinter import *
LED = 4; GPIO.setmode(GPIO.BCM); GPIO.setup(LED,GPIO.OUT)
```

```

def LedAllumée():
    GPIO.output(LED,True)

def LedÉteinte():
    GPIO.output(LED,False)

root = Tk(); root.title("LED")
Label(root,
      text="Veuillez cliquer sur le bouton, pour allumer et
      éteindre la LED").pack()
Button(root, text="Allumée", command=LedAllumée).pack(side=LEFT)
Button(root, text="Éteinte", command=LedÉteinte).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()

```



Fig. 10.2: Voilà à quoi la boîte de dialogue ressemble.

10.1.1 Voilà comment cela fonctionne

Ce programme présente les fonctions de base de la bibliothèque Tkinter pour construire la boîte de dialogue graphique. Contrairement à la bibliothèque graphique PyGame, qui construit les graphiques au pixel près, la taille de la boîte de dialogue et les éléments de contrôle dans Tkinter découlent automatiquement de la taille requise mais peuvent également être modifiés manuellement ultérieurement si nécessaire.

```

import RPi.GPIO as GPIO
from Tkinter import *

```

Après l'importation de la bibliothèque GPIO, les éléments de la bibliothèque Tkinter sont également importés.

```

LED = 4
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED,GPIO.OUT)

```

Ces lignes ne montrent rien de nouveau. Le port GPIO 4 est défini comme port de sortie pour une LED et désigné avec la variable LED.

```

def LedAllumée():
    GPIO.output(LED,True)

```

Une fonction LedAllumée() est maintenant définie. Elle allume la LED.

```

def LedÉteinte():
    GPIO.output(LED,False)

```

Une fonction semblable, LedÉteinte(), éteint la LED. Ces deux fonctions sont appelées plus tard à l'aide de deux boutons dans la boîte de dialogue.

Jusqu'à ce point, tout était en Python pur. Nous continuons maintenant avec T et des particularités.

`root = Tk()` Tkinter travaille avec ce qu'on appelle des widgets. Il s'agit d'éléments d'écran indépendants dans la plupart de cas pour contenir différents éléments de la boîte de dialogue. Chaque programme a besoin d'un widget `root` à partir duquel tous les autres objets sont appelés. Ce widget `root` est toujours nommé `Tk()`, génère automatiquement une fenêtre et initialise la bibliothèque Tkinter.

L'objet `root.title("LED")` dans Tkinter fournit différentes méthodes pour différentes fins. La méthode `title()` dans un widget fixe le titre de la fenêtre, écrit également dans ce cas le mot `LED` dans la ligne de titre de la nouvelle fenêtre.

Chaque widget peut contenir plusieurs objets qui sont définis séparément. Pour ce faire, Tkinter connaît différents types d'objet dont chacun des différents paramètres permettent d'écrire les propriétés de l'objet. Les paramètres sont spécifiés entre parenthèse, séparés par des virgules, derrière le type de l'objet. Étant donné que cette liste peut être très longue, on écrit généralement chaque paramètre sur une ligne distincte de sorte que tous les paramètres soient alignés les uns sous les autres. Contrairement à l'indentation des boucles et des requêtes dans Python, ces indentations ne sont pas obligatoires pour les objets Tkinter.

```
Label(root, text="Veuillez cliquer sur le bouton pour allumer et  
éteindre la LED").pack()
```

Les objets de type `Label` sont des textes bruts dans un widget. Ils peuvent être modifiés par le programme mais n'offrent aucune interaction avec l'utilisateur. Le premier paramètre dans chaque objet Tkinter est le nom du widget supérieur, la plupart du temps le nom de la fenêtre dans laquelle l'objet se trouve. Dans notre cas, il s'agit de la seule fenêtre dans le programme, le widget `root`.

Le paramètre `text` contient le texte qui doit être affiché sur le `Label`. À la fin de la définition de l'objet, le dit Conditionneur est inclus comme la méthode `.pack()`. Ce conditionneur intègre l'objet dans la boîte de dialogue et génère la géométrie du widget.

```
Button(root,  
        text="Allumé",  
        command=LedAllumée).pack(side=LEFT)
```

Les objets de type `Button` sont les boutons sur lesquels l'utilisateur clique pour déclencher une action donnée. Le paramètre `text` contient ici également le texte qui doit être affiché sur le bouton.

Le paramètre `command` contient une fonction qui appelle le bouton lorsque vous cliquez dessus. Aucun paramètre ne peut être passé et le nom de la fonction doit être donné sans parenthèse. Ce bouton appelle la fonction `LedÉteinte()` qui éteint la LED.

La méthode `.pack()` peut également contenir un autre paramètre qui spécifie comme un objet doit être disposé dans la boîte de dialogue. `side=LEFT` signifie que le bouton est aligné à gauche et non centré.

```
Button(root, text="Éteinte", command=LedÉteinte).pack(side=LEFT)
```

Un deuxième bouton est appliqué en suivant le même schéma. Il éteint la LED via la fonction `LedÉteinte()`.

Toutes les fonctions et tous les objets sont maintenant définis et le vrai programme peut être commencé.

`root.mainloop()` Le programme principal se compose uniquement d'une seule ligne. Elle lance la boucle principale `mainloop()`, une méthode du widget `root`. Cette boucle du programme attend que l'utilisateur actionne un des widget et déclenche ainsi une action.

Le symbole en haut à droite pour fermer la fenêtre n'a pas besoin d'être spécialement défini avec Tkinter. Si l'utilisateur ferme la fenêtre principale `root`, la boucle principale `mainloop()` est automatiquement quittée.

`GPIO.cleanup()` Le programme se poursuit jusqu'à la dernière ligne et ferme le port GPIO ouvert.

Après le démarrage du programme, une boîte de dialogue s'affiche à l'écran. Cliquez sur le bouton *Allumée*, pour allumer la LED, puis sur *Éteinte* pour l'éteindre à nouveau.

10.2 Contrôler le chenillard avec l'interface graphique

La bibliothèque Tkinter de Python met à disposition plusieurs éléments de commandes autres que les simples boutons. Les boutons radio permettent de construire les menus de sélection dans lesquels l'utilisateur peut choisir parmi plusieurs options données.

Qu'est-ce qu'un bouton radio ?

Le nom « bouton radio » vient en fait des vieilles radios où il y avait des boutons pour choisir parmi des stations de radio prédéfinies. À chaque fois que l'on appuyait sur l'un de ces boutons, le dernier bouton enfoncé était libéré automatiquement grâce à un mécanisme ingénieux. Les boutons radio se comportent de la même façon. Si l'utilisateur choisit une option, les autres sont automatiquement inactivées.

L'expérience suivante montre différents motifs clignotants de LED qui sont semblables à l'expérience « Motif à LED coloré et chenillard ». Contrairement à l'expérience précédente, l'utilisateur n'a pas besoin d'entrer son choix sur un écran textuel mais il peut choisir facilement le motif souhaité à partir d'une liste simple.

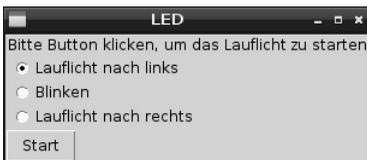


Fig. 10.3: La boîte de dialogue propose trois motifs à LED.

La construction du circuit est la même que dans l'expérience « Motif à LED coloré et chenillard ».

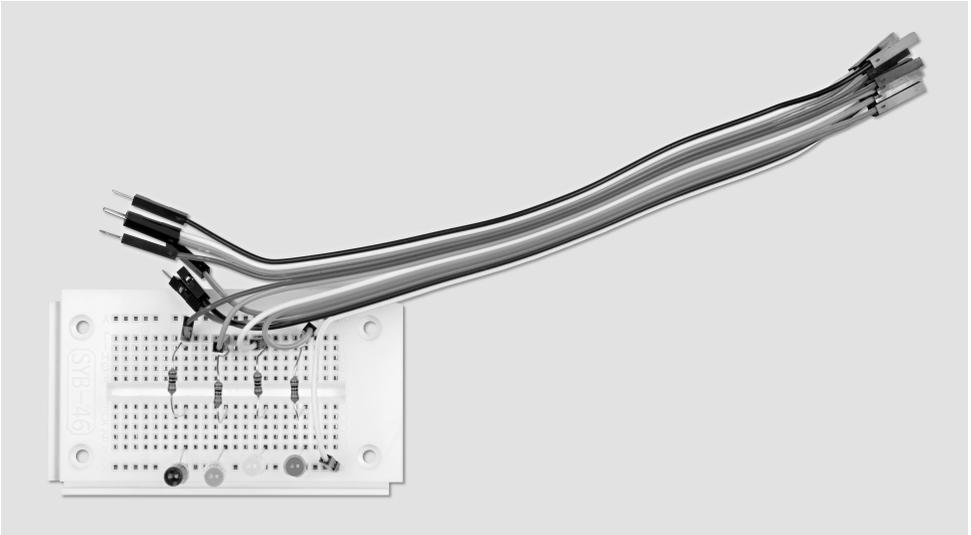
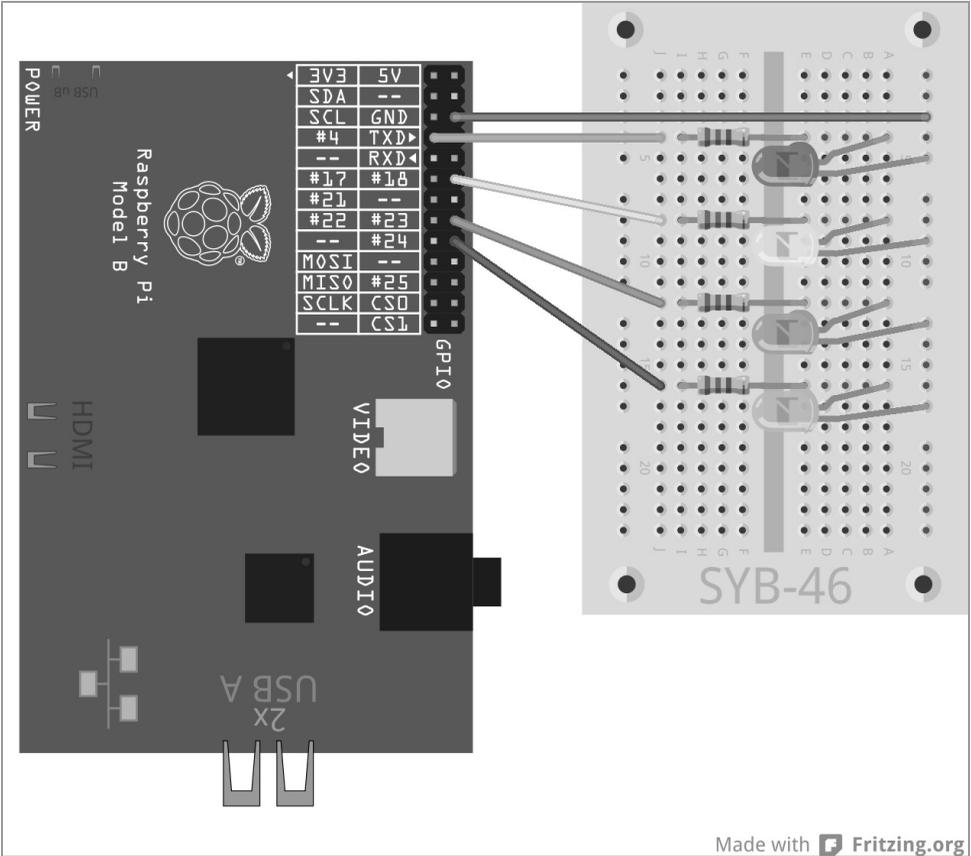


Fig. 10.4: Construction de la carte pour l'expérience 10.2

Composants nécessaires :
1x Carte de circuit imprimé
1x LED rouge
1x LED jaune
1x LED verte
1x LED bleue
4x Résistances de 220 Ω
5x Câbles de connexion



Made with  Fritzing.org

Fig. 10.5: Quatre LED clignotent avec des motifs différents.

Le programme `ledtk02.py` est basé sur le programme précédent mais les boutons radio et les fonctions pour le chenillard à LED et les motifs clignotant sont étendus.

```
import RPi.GPIO as GPIO
import time
from Tkinter import *
GPIO.setmode(GPIO.BCM)
LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)
w = 5; t = 0.2
motif = [
    ("Chenillard vers la gauche",1),
    ("Clignoter",2),
    ("Chenillard vers la droite",3)
```

```

]
root = Tk(); root.title("LED"); v = IntVar(); v.set(1)
def LedEin():
    e = v.get()
    if e == 1:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
                time.sleep(t)
                GPIO.output(LED[j], False)
    elif e == 2:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
                time.sleep(t)
            for j in range(4):
                GPIO.output(LED[j], False)
                time.sleep(t)
    else:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[3-j], True); time.sleep(t)
                GPIO.output(LED[3-j], False)
Label(root,
    text="Veuillez cliquer sur le bouton pour démarrer le
chenillard").pack()
for txt, m in motif:
    Bouton_radio(root, text = txt,
        variable = v, value = m).pack(anchor=W)
Button(root, text="Démarrer", command=LedAllumée).pack(side=LEFT)
root.mainloop()
GPIO.cleanup()

```

10.2.1 Voilà comment cela fonctionne

Les bibliothèques requises sont importées encore une fois au début. En outre par rapport au dernier programme, la bibliothèque `time` est également importée. Elle est nécessaire pour les temps d'attente des effets à LED clignotants.

```

GPIO.setmode(GPIO.BCM); LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)

```

Une liste est ensuite définie pour les quatre LED. Les ports GPIO correspondants sont définis comme des sorties et spécifiés sur 0 de sorte que toutes les LED soient éteintes au début.

```
w = 5; t = 0.2
```

Deux variables spécifient deux valeurs dans le programme : le nombre de répétition `w` d'un motif et le temps de clignotement `t`. Les deux valeurs peuvent être aussi inscrites à chaque occurrence dans le programme. De cette façon, elles peuvent s'ajuster plus facilement étant donné qu'elles ne sont définies qu'à un seul endroit.

```
motif = [
    ("Chenillard vers la gauche",1), ("Clignoter",2), ("Chenillard vers la droite",3)
]
```

Les textes des trois motifs disponibles sont définis sous forme de liste spécifique. Chacun des trois éléments de la liste consiste en quelques valeurs, chacun comprenant le texte affiché et la valeur numérique qui doit être retournée plus tard lors du choix du bouton radio.

```
root = Tk(); root.title("LED")
```

L'initialisation du widget `root` correspond encore une fois au programme précédent, ne différant qu'au niveau du contenu de la boîte de dialogue.

```
v = IntVar(); v.set(1)
```

Les variables qui sont utilisées dans la boîte de dialogue Tk, doivent être déclarées avant la première utilisation, contrairement aux variables normales de Python. Ces deux lignes déclarent une variable `v` comme un nombre entier et la spécifie comme `1` au début.

```
def LedAllumée():
    e = v.get()
```

Une fonction qui s'appelle `LedAllumée()` comme dans le dernier exemple est maintenant définie à nouveau. Mais cette fois, elle n'allume pas qu'une seule LED mais démarre un motif de LED. La deuxième fonction `LedÉteinte()` du dernier exemple n'est pas nécessaire ici. La première ligne de la nouvelle fonction lit l'entrée de l'utilisateur à partir de la variable tk `v` et écrit la valeur dans la variable Python `e`. Comme la valeur arrive exactement dans la variable `v`, apprenez davantage ci-dessous avec l'explication des boutons radio.

Selon le choix de l'utilisateur, trois différentes boucles de programme peuvent être lancées :

```
if e == 1:
    for i in range(w):
        for j in range(4):
            GPIO.output(LED[j], True); time.sleep(t)
            GPIO.output(LED[j], False)
```

Dans le premier cas, une boucle est exécutée cinq fois. Elle allume successivement chacune des quatre LED, les laisse allumées pendant 0,2 seconde puis les éteint. Les cinq répétitions et le temps de clignotement de 0,2 seconde sont définis par les variables `w` et `t` au début du programme.

```
elif e == 2:
    for i in range(w):
        for j in range(4):
            GPIO.output(LED[j], True)
            time.sleep(t)
        for j in range(4):
            GPIO.output(LED[j], False)
            time.sleep(t)
```

Dans le deuxième cas, toutes les quatre LED sont allumées à la suite simultanément, elles éclairent ensuite pendant 0,2 secondes puis sont éteintes en même temps.

```
else:
    for i in range(w):
        for j in range(4):
            GPIO.output(LED[3-j], True); time.sleep(t)
            GPIO.output(LED[3-j], False)
```

Le troisième cas ressemble au premier sauf que les LED sont comptées à l'envers et que, par conséquent, le chenillard court dans le sens inverse.

Après avoir défini la fonction, les éléments de l'interface graphique sont créés.

```
Label(root,
      text="Veuillez cliquer sur le bouton, pour démarrer le
      chenillard").pack()
```

Le texte de la boîte de dialogue est à nouveau défini comme un objet `Label`. La définition des trois boutons radio est nouvelle.

```
for txt, m in motif:
    Bouton_radio(root, text = txt, variable = v, value = m).pack(anchor=W)
```

Les boutons radio sont définis par une forme spéciale de la boucle `for`. A la place d'un compteur de boucle, il y a ici deux variables qui sont comptées en parallèle. Les deux compteurs passent successivement par les éléments de la liste `motif`. La première variable de comptage `txt` assume la première valeur de la paire de valeurs : le texte affiché à côté du bouton radio. La deuxième variable de comptage `m` assume le numéro de chaque motif à partir de la deuxième valeur de la paire de valeur.

La boucle crée de cette façon trois boutons radio dont le premier paramètre est toujours le widget `root` où se trouvent les boutons radio. Le paramètre `text` d'un bouton radio donne le texte à afficher qui est dans ce cas lu à partir de la variable `txt`. Le paramètre `variable` spécifie une variable Tk précédemment déclarée, dans laquelle la valeur du bouton radio choisi par l'utilisateur est enregistrée.

Le paramètre `value` spécifie une valeur numérique pour chaque bouton radio, qui est lue dans ce cas à partir de la variable `m`. Si un utilisateur clique sur ce bouton radio, la valeur du paramètre `value` dans la variable enregistrée sous `variable` est écrite. Chacun des trois boutons radio est construit selon sa définition de manière similaire à la méthode `.pack()` dans la boîte de dialogue. Le paramètre `anchor=W` assure que les boutons radio soient alignés à gauche les uns sous les autres.

```
Button(root, text="Démarrer", command=LedEin).pack(side=LEFT)
```

Le bouton est défini comme dans le dernier exemple.

```
root.mainloop(); GPIO.cleanup()
```

La boucle principale et la fin du programme correspondent à l'exemple précédent.

Lancez le programme et choisissez un motif de LED via une des boutons radio. Le premier choix est présélectionné via la variable `v`. Si vous utilisez les boutons radio dans une boîte de dialogue, vous devez toujours définir une présélection judicieuse pour ne jamais obtenir de résultat indéfini si l'utilisateur ne fait aucun choix. En cliquant sur *Start*, vous lancez le motif choisi et il s'exécute cinq fois. Vous pouvez ensuite choisir un autre motif.

10.3 Régler la vitesse de clignotement

La boîte de dialogue est étendue dans la troisième étape. L'utilisateur peut maintenant régler la vitesse du clignotement à l'aide d'un curseur.

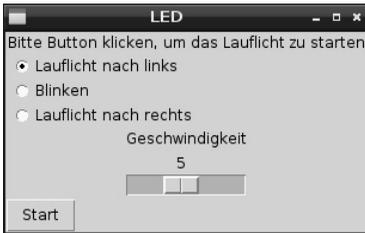


Fig. 10.6: Trois motifs de LED au choix et vitesse de clignotement réglable

Utilisation du curseur

Le curseur offre une manière très intuitive d'entrer des valeurs numériques comprises dans une certaine gamme. De cette façon, on économise une requête de plausibilité qui détermine si l'utilisateur a donné une valeur que le programme peut appliquer judicieusement, étant donné que les valeurs en dehors de la gamme prédéfinie du curseur ne sont pas possibles. Réglez toujours le curseur de sorte que la valeur soit concevable pour l'utilisateur. Cela n'a aucun sens de pouvoir régler les curseurs dans une fourchette de plusieurs millions. Si la valeur numérique absolue ne joue aucun rôle réel, il vous suffit de donner à l'utilisateur une échelle de 1 à 10 ou de 1 à 100 et de calculer la valeur dans le programme. La valeur doit être augmentée de gauche à droite. Si cela se fait dans le sens inverse, cela sera étrange pour la plupart des utilisateurs. En outre, prédéfinissez toujours une valeur judicieuse qui est appliquée si l'utilisateur ne change pas le curseur.

Le programme `ledtk03.py` correspond en grande partie à l'exemple précédent. Seule la régulation de la vitesse est ajoutée.

```
import RPi.GPIO as GPIO
import time
from Tkinter import *

GPIO.setmode(GPIO.BCM); LED = [4,18,23,24]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=0)

w = 5
motif = [
    ("Chenillard vers la gauche",1), ("Clignoter",2), ("Chenillard vers la droite",3)
```

```

]
root = Tk(); root.title("LED"); v = IntVar(); v.set(1); g = IntVar(); g.set(5)
def LedEin():
    e = v.get()
    t = 1.0/g.get()
    if e == 1:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True); time.sleep(t)
                GPIO.output(LED[j], False)
    elif e == 2:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[j], True)
                time.sleep(t)
            for j in range(4):
                GPIO.output(LED[j], False)
                time.sleep(t)
    else:
        for i in range(w):
            for j in range(4):
                GPIO.output(LED[3-j], True): time.sleep(t)
                GPIO.output(LED[3-j], False)
Label(root,
    text="Veuillez cliquer sur le bouton pour démarrer le
chenillard").pack()
for txt, m in motif:
    Bouton_radio(root, text = txt, variable = v, value = m).pack(anchor=W)
Label(root, text="Vitesse").pack()
Scale(root, orient=HORIZONTAL, from_ = 1, to = 10, variable = g).pack()
Button(root, text="Démarrer", command=LedAllumée).pack(side=LEFT)

root.mainloop()
GPIO.cleanup()

```

10.3.1 Voilà comment cela fonctionne

L'initialisation des bibliothèques et des ports GPIO, ainsi que la définition de la liste pour les trois motifs clignotants de LED correspondent au programme précédent. La définition de la variable `t` pour le temps de clignotement disparaît puisqu'elle est lue ultérieurement sur le curseur.

`g = IntVar(); g.set(5)` En plus de la variable Tk `v`, dans laquelle le motif clignotante choisi est enregistré, une autre variable entière `g` est déclarée pour la vitesse. Elle contient une valeur de départ de 5, qui correspond à la valeur centrale du curseur.

```

def LedAllumée():
    e = v.get(); t = 1.0/g.get()

```

La fonction qui permet de faire clignoter les LED correspond aussi à l'exemple précédent, avec cependant une différence. La variable `t` pour la durée de clignotement est déterminée à partir de la valeur `g` du curseur.

Étant donné qu'un utilisateur fait le lien intuitivement en un clin d'œil avec une vitesse plus élevée, le curseur placé à droite retourne une valeur plus élevée. Cependant, dans le programme, un temps d'attente plus court (donc une valeur plus petite) est nécessaire avec une vitesse plus élevée. Cela est obtenu par un calcul inverse, qui détermine à partir des valeurs 1 à 10 du curseur la valeur 1,0 à 0,1 pour la variable `t`. Dans la formule, on doit écrire 1,0 et non 1 pour que le résultat soit un chiffre à virgule et non un chiffre entier.

Convertir les chiffres entiers en chiffre à virgule

Le résultat d'un calcul est automatiquement enregistré comme un chiffre à virgule si au moins une des valeurs est un chiffre à virgule. Si toutes les valeurs sont exprimées en nombres entiers (Integer), le résultat est alors un nombre entier.

La définition du label et des boutons radio dans la boîte de dialogue sont reprises à partir de l'exemple précédent.

```
Label(root,  
      text="Vitesse").pack()
```

Pour expliquer le curseur; un autre label est écrit dans la boîte de dialogue. Étant donné qu'il ne contient aucun paramètre dans la méthode `pack()`, il est centré horizontalement sous le bouton radio.

```
Scale(root, orient=HORIZONTAL, from_ = 1, to = 10, variable = g).pack()
```

Le curseur est un objet de type `Scale`, qui a comme premier paramètre `root` comme tous les objets dans cette boîte de dialogue. Le paramètre `orient=HORIZONTAL` indique que le curseur est placé à l'horizontale. Il serait placé à la verticale sans ce paramètre. Les paramètres `from_` et `to` indiquent le début et la fin du curseur. Faites attention à l'écriture de `from_`, car `from` sans tiret bas est un mot réservé dans Python pour l'importation des bibliothèques. Le paramètre `variable` spécifie une variable Tk précédemment déclarée, dans laquelle la valeur actuellement réglée sur le curseur est enregistrée. La valeur de départ est reprise à partir de la valeur fixée par la déclaration des variables, dans ce cas 5.

Le curseur est également centré horizontalement dans la boîte de dialogue avec la méthode `pack()`.

Les autres parties du programme - le bouton *Start*, la boucle principale et la fin du programme - ne changent pas par rapport à l'exemple précédent.

Démarrez le programme, choisissez un motif clignotant et réglez la vitesse. Plus la valeur est élevée, plus le motif clignote rapidement. En cliquant sur le bouton *Start*, la fonction `LedAllumée()` détermine le motif clignotant choisi à partir des boutons radio ainsi que la vitesse à partir de la position du curseur.

11 Pitance avec les LED

À la fin des années 1970, avant la vraie époque des jeux informatiques, il y avait un jeu électronique avec quatre ampoules de couleur, qui a même été nominé pour le jeu de l'année en 1979. Le jeu était vendu en Allemagne sous le nom *Senso*. Atari l'a reproduit avec un jeu nommé *Touch Me* de la taille d'une grosse calculatrice. Un autre jeu semblable est sorti sous le nom *Einstein*. *Senso* a été commercialisé sous le nom *Simon* pour sa version anglophone.

Raspbian fournit une version graphique de ce jeu avec les *Python Games*, sous le nom *Simulate*.

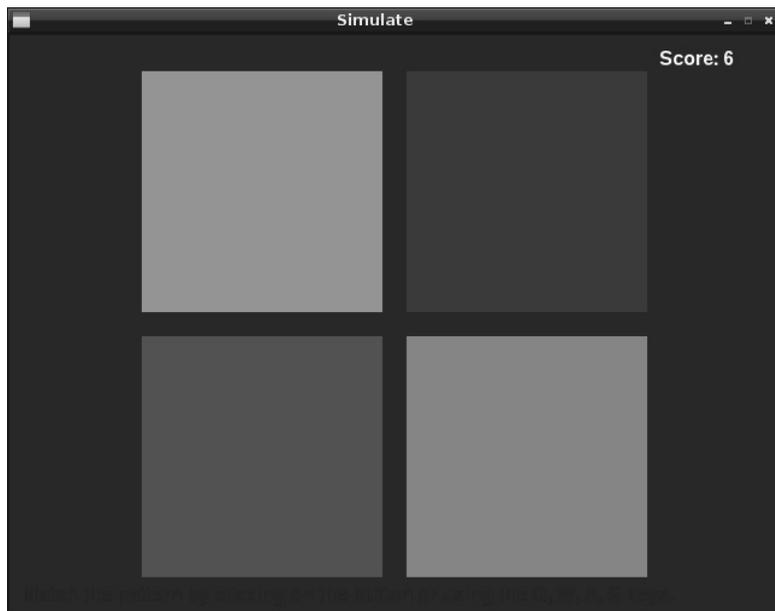


Fig. 11.1: Le jeu Simulate tiré des Python Games.

Notre jeu PiDance est également basé sur ce principe de jeu. Les LED clignotent dans un ordre aléatoire. L'utilisateur doit ensuite appuyer sur les boutons dans le même ordre. À chaque tour, une LED supplémentaire s'allume de sorte qu'il soit toujours plus difficile de se rappeler la séquence des LED. Dès que l'on fait une erreur, le jeu est terminé.

Le jeu est construit à partir de deux cartes de circuit imprimé, de sorte que les touches soient sur le bord et faciles à utiliser, sans que des câbles ne soient débranchés accidentellement des cartes de circuit imprimé. Pour une meilleure stabilité, les cartes de circuit imprimé peuvent être interconnectées longitudinalement.

En plus des câbles de connexion que vous connaissez déjà, quatre cavaliers courts sont également nécessaires. Pour ce faire, coupez avec un pince coupante ou un coupe-fil le fil de connexion en des morceaux longs de 2,5 centimètres environ et dénudez 7 millimètres de gaine isolantes au niveau des deux

extrémités à l'aide d'une lame coupante. Pliez ces morceaux de fil pour obtenir une forme en U. Vous pouvez ensuite connecter deux rangées sur une carte de circuit imprimé.

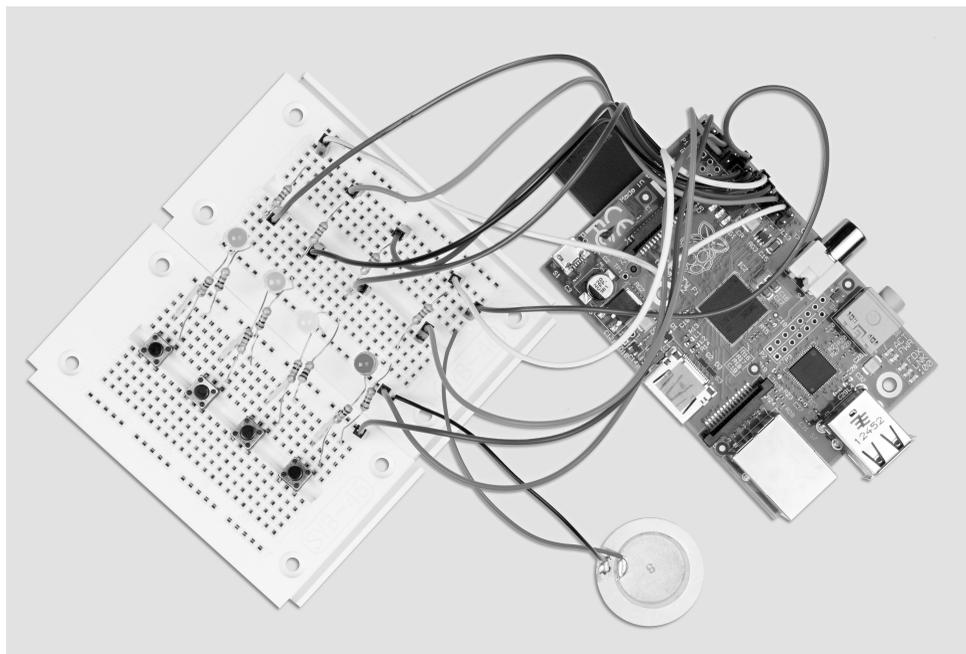


Fig. 11.2: Construction de la carte pour l'expérience 11

Composants nécessaires :

2x Cartes de circuit imprimé

1x LED rouge

1x LED jaune

1x LED verte

1x LED bleue

4x Résistances de 220Ω

4x Résistances de $1 \text{ k}\Omega$

4x Résistances de $10 \text{ k}\Omega$

4x Boutons

10x Câbles de connexion

4x Cavaliers courts

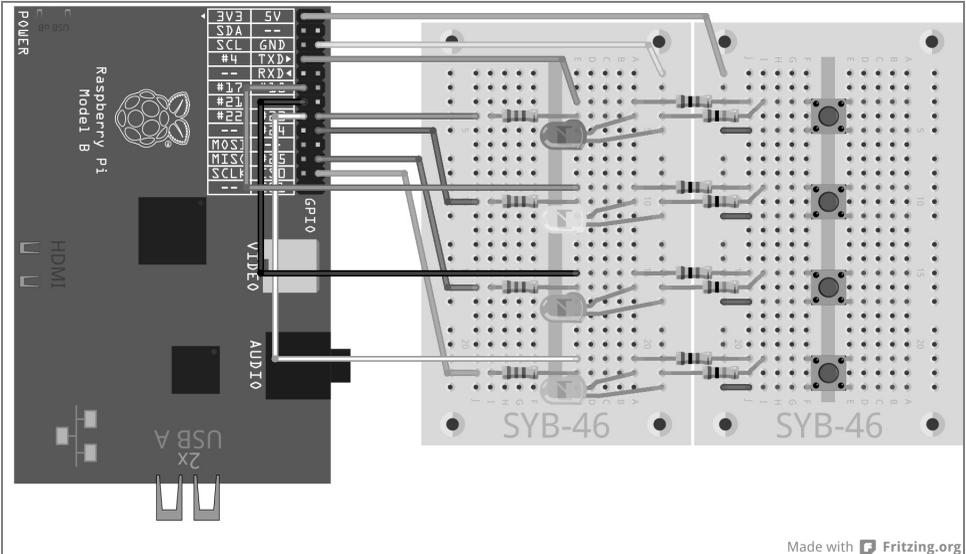


Fig. 11.3: PiDance avec LED et des boutons sur les deux cartes de circuit imprimé

Les boutons sont montés à côté des LED associées. Les deux rangées longitudinales centrales des cartes de circuit imprimé des deux côté du point de jonction servent de conducteur 0 V et + 3,3 V pour le circuit.

Le programme `pidance01.py` contient le jeu terminé.

```
# -*- coding: utf-8 -*-
import time, random
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
rchiffre = 10; couleur = []
for i in range(rchiffre):
    Couleur.append(random.randrange(4))
LED = [23,24,25,8]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
BOUT = [4,17,21,22]
for i in BOUT:
    GPIO.setup(i, GPIO.IN)
def LEDAllumée(n, z):
    GPIO.output(LED[n], True); time.sleep(z)
    GPIO.output(LED[n], False); time.sleep(0.15)
def Appuyer():
    while True:
        if(GPIO.input(BOUT[0])):
            return 0
        if(GPIO.input(BOUT[1])):
            return 1
        if(GPIO.input(BOUT[2])):
```

```

        return 2
    if(GPIO.input(BOUT[3])):
        return 3
ok = True
for tour in range(1, rchiffre +1):
    print "Tour", tour
    for i in range(tour):
        LEDallumée(couleur[i], 1)
    for i in range(tour):
        bouton = Appuyer()
        LEDallumée(bouton, 0.2)
        if(bouton != couleur[i]):
            print "Perdu !"
            print "Vous avez réussi jusqu'au tour ", tour - 1, "."
            for j in range(4):
                GPIO.output(LED[j], True)
            for j in range(4):
                time.sleep(0.5)
                GPIO.output(LED[j], False)
            ok = False
            break
    if(ok == False):
        break
    time.sleep(0.5)
if(ok == True):
    print "Bien joué !"
    for i in range(5):
        for j in range(4):
            GPIO.output(LED[j], True)
        time.sleep(0.05)
        for j in range(4):
            GPIO.output(LED[j], False)
        time.sleep(0.05)
GPIO.cleanup()

```

11.1.1 Voilà comment cela fonctionne

Le programme propose de nombreuses nouveautés. Les bases du contrôle GPIO sont cependant déjà connues.

`rchiffre = 10` Après l'importation des modules `time`, `random` et `RPi.GPIO`, une variable `rchiffre` est créée. Elle spécifie le nombre de tours du jeu. Vous pouvez évidemment jouer à plus de dix tours - plus il y a de tours, plus il est difficile de se rappeler de la séquence de clignotement.

```

couleur = []
for i in range(rchiffre):
    couleur.append(random.randrange(4))

```

La liste `couleur` est remplie avec une boucle avec autant de nombres aléatoires compris entre 0 et 3 que de tours joués. Pour ce faire, la méthode `append()` est utilisée. Elle indique ce qui est disponible dans chaque liste. Elle dépend des éléments définis comme paramètre sur la liste.

```
LED = [23,24,25,8]
for i in LED:
    GPIO.setup(i, GPIO.OUT, initial=False)
```

Les ports GPIO pour les LED sont configurés comme des sorties selon le schéma connu dans une liste LED et ils sont tous inactivés.

```
BOUT = [4,17,21,22]
for i in BOUT:
    GPIO.setup(i, GPIO.IN)
```

Suivant le même principe, les ports GPIO sont configurés comme des entrées pour les quatre boutons dans une liste BOUT.

Les bases sont ainsi créées et elles définissent encore deux fonctions qui sont nécessaires plusieurs fois dans le programme.

```
def LEDallumée(n, z):
    GPIO.output(LED[n], True); time.sleep(z)
    GPIO.output(LED[n], False); time.sleep(0.15)
```

La fonction LEDallumée() allume une LED qui éclaire alors pendant un certain temps. La fonction utilise deux paramètres. Le premier paramètre n donne le numéro de la LED compris entre 0 et 3, le deuxième paramètre z donne le temps pendant lequel la LED doit rester allumée. Après que la LED est éteinte à nouveau, la fonction attend encore que 0,15 seconde se soit écoulée, pour voir les pauses courtes entre les allumages des LED en cas d'appels multiples. Cela est particulièrement important lorsqu'une LED est allumée plusieurs fois à la suite. Autrement, on ne pourrait pas s'en apercevoir.

```
def Appuyer():
    while True:
        if(GPIO.input(BOUT[0])):
            return 0
        if(GPIO.input(BOUT[1])):
            return 1
        if(GPIO.input(BOUT[2])):
            return 2
        if(GPIO.input(BOUT[3])):
            return 3
```

La fonction Appuyer() se compose d'une boucle infinie qui attend que l'utilisateur appuie sur un des boutons. Le numéro du bouton est ensuite transmis au programme principal.

ok = True Après la définition des fonctions, le vrai programme principal démarre et spécifie en premier une variable ok sur True. Dès que le joueur fait une erreur, ok passe à False. Si la variable est toujours True après le nombre de tour prédéfini, le joueur a gagné la partie.

```
for tour in range(1, rchiffre +1):
```

Le jeu exécute le nombre de tours fixé dans la variable rzahl. Le compteur de tours est décalé de 1 vers le haut pour que le jeu commence au tour 1 et non au tour 0.

```
print "Tour", tour
```

Le tour actuel est indiqué dans la fenêtre Python-Shell.

```
for i in range(tour):  
    LEDallumée(couleur[i], 1)
```

Le programme exécute maintenant le motif que le joueur doit se rappeler. Selon le nombre actuel de tours, plusieurs LED s'allument successivement en fonction de la liste `couleur` spécifiée au début du programme, avec des couleurs choisies aléatoirement. Comme le compteur `tour` commence à 1, une seule LED s'allume déjà pendant le premier tour. Pour allumer les LED, la fonction `LEDallumée()` est utilisée. Son premier paramètre est la couleur à partir de la position correspondante dans la liste. Son deuxième paramètre laisse chaque LED allumée pendant une seconde.

`for i in range(tour):` Après l'exécution du motif de couleur, une nouvelle boucle démarre dans laquelle le joueur doit entrer le même motif de mémoire, à l'aide des boutons.

`bouton = Appuyer()` Pour ce faire, la fonction `Appuyer()` est appelée. Elle attend jusqu'à ce que le joueur appuie un des boutons. Le numéro des boutons appuyés est enregistré dans la variable `bouton`.

`LEDallumée(bouton, 0.2)` Après que vous avez appuyé sur un bouton, la LED correspondante s'allume brièvement pendant 0,2 seconde.

`if(bouton != couleur[i]):` Si le dernier bouton enfoncé ne correspond pas à la couleur de la position correspondante dans la liste, le joueur a perdu. L'opérateur `!=` signifie non égal. L'opérateur `<>` peut aussi être utilisé ici.

```
print "Perdu !"  
print "Vous avez réussi jusqu'au tour ", tour - 1, "."
```

Le programme affiche sur l'écran que le joueur a perdu et combien de tours il a gagnés. Le nombre de tours passés est plus petit de 1 comme le compteur de tours actuel.

```
for j in range(4):  
    GPIO.output(LED[j], True)
```

Toutes les LED sont allumées comme signal visuel ...

```
for j in range(4):  
    time.sleep(0.5); GPIO.output(LED[j], False)
```

... puis elles sont éteintes les unes après les autres avec un intervalle de 0,5 secondes. Cela donne un effet de réduction visible.

`ok = False` La variable `ok`, qui indique si le joueur joue encore, est spécifié sur `False` ...

`break` ... et la boucle est interrompue. Le joueur ne peut plus appuyer sur un autre bouton. Le jeu est quitté immédiatement à la première erreur.

```
if(ok == False):  
    break
```

si `ok` est sur `False`, la boucle externe est également interrompue et il n'y a pas de tours supplémentaires.

`time.sleep(0.5)` Si les entrées ont répété correctement la séquence, le programme attend pendant 0,5 seconde jusqu'à ce que le tour suivant commence.

`if(ok == True):` Le programme arrive à cet endroit si les boucles se sont toutes exécutées, si le joueur a entré correctement toutes les séquences, ou si la boucle précédente a été interrompue par une erreur du jeu. Si `ok` reste sur `True`, la remise des prix a lieu. Sinon ce bloc est ignoré et le jeu exécute seulement la dernière ligne du programme.

```
print "Bien joué !"  
for i in range(5):  
    for j in range(4):  
        GPIO.output(LED[j], True)  
        time.sleep(0.05)  
    for j in range(4):  
        GPIO.output(LED[j], False)  
        time.sleep(0.05)
```

En cas de victoire, un affichage apparaît dans la fenêtre Python-Shell. Après cela toutes les LED clignotent brièvement cinq fois à la suite.

`GPIO.cleanup()` La dernière ligne est exécutée dans tous les cas. Tous les ports GPIO utilisés sont ici fermés.

Mentions légales

© 2014 Franzis Verlag GmbH, Richard-Reitzner-Allee 2, 85540 Haar bei München

www.elo-web.de

Auteur : Christian Immler

ISBN 978-3-645-10145-5

Tous droits réservés, y compris en ce qui concerne la reproduction photomécanique et le stockage sur les supports électroniques. La création et la distribution de copies sur papier, sur des supports de données ou sur Internet, en particulier dans le format PDF, n'est autorisée qu'avec la permission expresse de l'éditeur, sous peine de poursuites.

La plupart des noms de produits du matériel informatique et des logiciels, ainsi que les noms de marque et logos de société, qui sont mentionnés dans ce document, sont également et généralement des marques déposées et ils doivent être traités comme tels. Lorsque l'éditeur mentionne des noms de produits, il respecte en principe la graphie du fabricant.

Tous les circuits et programmes présentés dans ce livre ont été développés, vérifiés et testés avec le plus grand soin. Néanmoins, nous ne pouvons pas exclure complètement la présence d'erreur dans ce livre et dans le logiciel. L'éditeur et l'auteur sont responsables en cas d'intention délictueuse ou de négligence grave conformément aux dispositions légales. Par ailleurs, l'éditeur et l'auteur dans le cadre de la loi sur la responsabilité civile du producteur, en cas d'atteinte à la vie, à l'intégrité physique ou à la santé de la personne, ainsi qu'en cas de manquement fautif et essentiel aux obligations du contrat. Le droit aux dommages et intérêts pour avoir manqué aux obligations contractuelles importantes est cependant limité aux dommages spécifiques au contrat, sauf dans les cas où une responsabilité légale est fournie conformément à la loi sur la responsabilité du produit.



Les appareils électriques et électroniques ne doivent pas être jetés avec les ordures ménagères !

Il convient de procéder à l'élimination du produit au terme de sa durée de vie conformément aux prescriptions légales en vigueur. Il existe des points de collecte où vous pouvez déposer gratuitement vos appareils électriques usagés. Contactez votre commune pour savoir où se trouvent ces points de collecte.



Ce produit est conforme aux directives CE pertinents, tant que vous l'utilisez en respectant les instructions fournies. Ce mode d'emploi fait partie du produit et doit être transmis avec le produit si vous le cédez.

Attention ! Protection des yeux et LED.

Ne regardez jamais directement dans une LED car cela peut endommager la rétine ! Cela est particulièrement vrai pour les LED lumineuses dans des boîtiers clairs et en particulier pour les LED très puissantes. La luminosité apparente des LED blanches, bleues, violettes et à ultraviolets donne une fausse impression du risque réelle que faites prendre à vos yeux. Faites particulièrement attention si vous utilisez des lentilles convergentes. Utilisez les LED comme prévu dans les instructions mais pas avec des courants plus élevés.