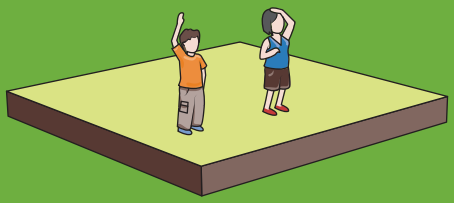
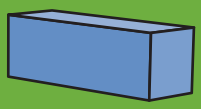
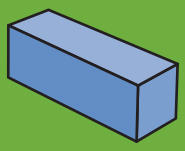






Chapitre 10 bonus

L'ascenseur Minecraft

DANS LE CHAPITRE 5, tu as appris à connecter des circuits électroniques à Minecraft, à faire clignoter des LED et à appuyer sur des boutons pour déclencher des événements dans le jeu. Il t'est déjà probablement venu à l'esprit que tu pouvais remplacer le bouton et la LED par n'importe quel autre type d'objet du monde réel.

Dans ce chapitre bonus, tu vas t'intéresser de plus près à cette idée de relier le monde virtuel de Minecraft au monde réel grâce à l'électronique. Pour cela, tu vas construire un ascenseur Minecraft qui pourra t'élever très haut dans le ciel depuis les plus obscurs abîmes. Sur la figure 10-1, tu peux voir les étages et le plateau de l'ascenseur Minecraft. C'est un ascenseur très simple et totalement dégarni, sans mur ni quoi que ce soit d'autre autour. Le plateau est la dalle en pierre qui te conduira à tous les niveaux du monde de Minecraft. Mais outre sa simplicité apparente, cet ascenseur possédera une caractéristique bien spéciale : tu vas en effet lui confectionner un panneau de commandes réel réalisé à partir de composants électroniques.

Comme tu es désormais un expert de la programmation dans Minecraft, nous avons volontairement élevé le niveau de difficulté de ce chapitre bonus. Il est disponible gratuitement sur le site Internet d'Eyrolles (`www.editions-eyrolles.com/dl/14292`). Mais si tu n'as pas encore acheté notre livre, nous te conseillons vivement de le faire et de réaliser tous les projets des chapitres 1 à 9, car ce chapitre bonus fait appel à toutes les connaissances qui y sont enseignées. Tu en apprendras bien davantage sur

la programmation dans Minecraft si tu les acquiers étape par étape avant de te lancer dans ce chapitre bonus.

L'ascenseur que tu vas construire est très épuré. Il ne comporte aucune paroi ni aucune porte, ne dispose pas d'une vraie cage d'ascenseur et les étages sont invisibles. Nous l'avons fait exprès, pour que tu puisses l'intégrer facilement à tes autres programmes une fois que tu l'auras terminé. C'est d'ailleurs ainsi que les choses se passent dans la vraie vie : les constructeurs d'ascenseurs conçoivent et construisent des ascenseurs et les bâtisseurs d'immeubles conçoivent et bâtissent des immeubles !

FIGURE 10-1 L'ascenseur Minecraft en état de fonctionnement, avec son plateau de cabine et ses différents étages

Ce dont tu as besoin

Normalement, tu disposes déjà de tous les composants nécessaires pour mener à bien ce chapitre. Tu peux les configurer en suivant les instructions du début du chapitre 5 ou, si tu as suivi mes instructions de la fin du chapitre, tes composants sont déjà connectés comme il le faut à la plaque d'essais. Dans ce chapitre, tu vas utiliser le même circuit électronique que tu as monté pour les besoins du chapitre 5, mais cette fois-ci tu vas utiliser quatre boutons au lieu d'un. Si tu l'as démonté, il vaudra mieux que tu relises les instructions du chapitre 5 pour le réassembler avant de continuer ce chapitre. Cela te fera gagner beaucoup de temps.

Construire l'ascenseur Minecraft

L'ascenseur Minecraft est un ascenseur entièrement fonctionnel qui oscille entre plusieurs étages. Ici, tu vas lui créer quatre niveaux. Il sera doté d'un panneau de commandes avec quatre boutons correspondant aux étages et un afficheur 7 segments qui indiquera l'étage auquel il se trouve. Chaque étage sera en outre doté d'une rangée de blocs actifs que tu devras frapper pour appeler l'ascenseur. Mais ce n'est pas tout, ton ascenseur sera également intelligent ! Il transportera ton joueur d'un étage à l'autre seulement si celui-ci se trouve à l'intérieur, pas s'il se trouve à l'extérieur.

Une fois terminé, l'ascenseur te servira de base dans la construction de tes immeubles. Plus tard, tu pourras lui ajouter autant d'étages que tu disposeras de broches GPIO sur ton circuit.

Si tu utilises un PC ou un Mac, tu dois te procurer un circuit Arduino Pro Micro présoudé. Et tu dois utiliser le kit anyio ainsi que le pilote `seg.py`, tous deux fournis dans le kit de démarrage et disponibles en téléchargement sur ma page GitHub : `https://github.com/whaleygeek/anyio`.

Compléter la liste des composants électroniques pour l'ascenseur

L'ascenseur Minecraft que tu vas construire sera doté de quatre boutons, pour t'amener aux quatre étages de ton futur immeuble. Pour l'instant, ton circuit ne comporte qu'un seul bouton (pour le premier étage), alors tu vas devoir en ajouter d'autres.

1. Place trois nouveaux boutons-poussoirs sur un espace libre de ta plaque en enfonçant délicatement les broches, comme tu l'as fait dans le chapitre 5, et en t'assurant de les répartir sur le milieu de la plaque d'essais. Pour cela, veille à brancher les deux broches supérieures du bouton sur la partie moyenne supérieure de la plaque et les deux broches inférieures à la partie moyenne inférieure de la plaque. Souviens-toi que lorsque tu appuies sur un bouton, il referme le circuit au niveau des broches de gauche et de droite.
2. Ajoute trois autres résistances de 10 Kohm (code couleur : marron, noir, orange). Pour cela, connecte l'une des pattes de ta résistance sur la bande positive (3,3 volts) située en haut de la plaque. N'oublie pas que ces résistances de tirage sont nécessaires pour garantir que la tension des broches GPIO se situe toujours entre 0 et 3,3 volts, te préservant ainsi des erreurs de manipulation avec les boutons.

3. Place trois fils supplémentaires sur la bande de 0 volt de la plaque et relie leurs extrémités aux broches inférieures droites de tes boutons, afin de connecter les bonnes broches GPIO à la bande négative lorsque tu appuies sur un bouton.
4. Place trois fils supplémentaires sur les broches supérieures gauches des boutons et connecte leurs extrémités aux broches GPIO suivantes :

Pour Raspberry Pi

- Étage 2 = GPIO 14
- Étage 3 = GPIO 23
- Étage 4 = GPIO 24

Pour Arduino

- Étage 2 = GPIO 5
- Étage 3 = GPIO 2
- Étage 4 = GPIO 3

Tu as terminé ? Compare tes branchements à ceux de la figure 10-2 si tu as un Raspberry Pi et à ceux que la figure 10-3 si tu as un circuit Arduino.

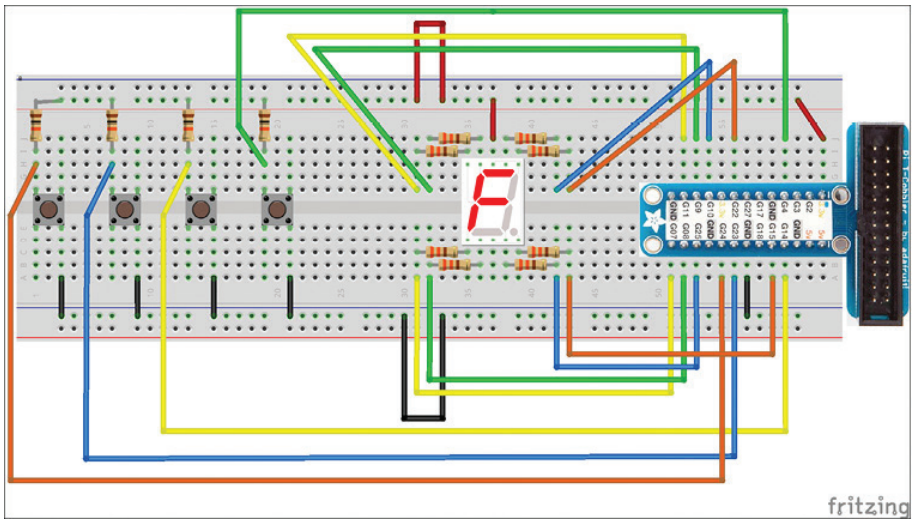


FIGURE 10-2 Configuration du panneau de commandes sur Raspberry Pi

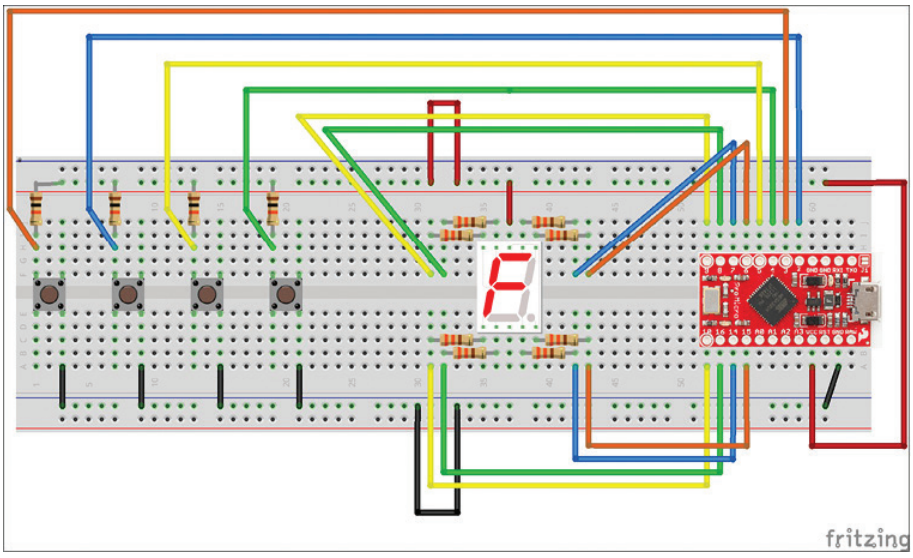


FIGURE 10-3 Configuration du panneau de commandes sur Arduino

Écrire la structure du programme

Comme le programme de contrôle de l'ascenseur va être assez long, tu vas devoir le développer et le tester en plusieurs parties. À cette étape du chapitre, ton programme principal ne réalisera pas grand-chose, mais tu y ajouteras des fonctionnalités petit à petit, comme tu l'as fait dans les chapitres précédents du livre. N'oublie pas que si tu as acheté un afficheur 7 segments différent de celui du livre, tu devras peut-être modifier la constante `ON` et lui affecter la valeur `True` s'il s'agit d'un afficheur à cathode commune. Pour ma part, j'ai utilisé un afficheur à anode commune. Retourne au chapitre 5 si jamais tu as un doute.

1. Crée un nouveau fichier en cliquant sur `File>New File` et nomme-le `ascenseur.py`.
2. Importe les modules dont tu as besoin. Tu utiliseras le même pilote `seg7.py` que dans les chapitres précédents pour prendre le contrôle de ton afficheur 7 segments :

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import time
import anyio.seg7 as display
```

3. Importe ensuite le module GPIO correspondant à ton ordinateur et attribue les bons numéros de broche GPIO aux broches que tu vas utiliser. Ces deux constantes GPIO sont des listes Python. Cela te facilitera la tâche ensuite lorsque tu les utiliseras dans des boucles, car cela raccourcira ton programme et le simplifiera :

Sur Raspberry Pi

```
import RPi.GPIO as GPIO
ETAGE_GPIO = [4, 14, 23, 24] # il est important de respecter
                             # cet ordre
AFFICHEUR_GPIO = [10,22,25,8,7,9,11,15] # il est important de
                                        # respecter cet ordre
ON = False # False=anode commune, True=cathode commune
```

Sur Arduino

```
import anyio.GPIO as GPIO
ETAGE_GPIO = [4, 5, 2, 3] # il est important de respecter
                          # cet ordre
AFFICHEUR_GPIO = [7,6,14,16,10,8,9,15] # il est important de
                                       # respecter cet ordre
ON = False # False=anode commune, True=cathode commune
```

4. Définis maintenant quelques constantes pour ajouter des informations sur les étages. Cela t'aidera plus tard si tu décides d'en ajouter ou d'adapter leur hauteur à ton immeuble :

```
NOMBRE_ETAGES = len(ETAGE_GPIO) # Il y a autant d'étages que
                                # de boutons
HAUTEUR_ETAGE = 10
```

5. Ton ascenseur ne peut répondre qu'à l'un de ces trois états : arrêté, descendant ou ascendant. Tu dois donc en faire des constantes. Cela simplifiera grandement la logique, la lecture et la modification de ton programme :

```
ARRET = 0
DESCENTE = 1
ASCENSION = 2
```

6. Pose ensuite quelques variables pour que l'ascenseur soit toujours construit à côté de la position du joueur lorsque le programme s'exécute :

```
mc = minecraft.Minecraft.create()
pos = mc.player.getTilePos()
ASCENSEUR_X = pos.x+3
ASCENSEUR_Y = pos.y
ASCENSEUR_Z = pos.z+3
etat_ascenseur = ARRET
ascenseur_y = ASCENSEUR_Y
```

7. Crée une liste Python pour enregistrer les demandes d'arrêt. Ton ascenseur utilisera ces variables booléennes dès lors que tu appuieras sur l'un des boutons du dispositif ou que tu frapperas l'un des blocs de diamant situés aux différents étages. Le programme de l'ascenseur s'arrêtera alors à tous les étages dont la

variable est définie sur `True` dans la liste. Pour en savoir plus sur cette liste, consulte l'encadré « Percer les secrets du code ».

```
demandes_enregistrees = [False, False, False, False]
```

8. Insère du code factice en dessous de toutes tes fonctions. Tu les compléteras plus tard au fur et à mesure que tu développeras et testeras chacune des fonctionnalités de ton ascenseur :

```
def creerAscenseur():
    print("creerAscenseur")

def tracerAscenseur(y, blockid):
    print("tracerAscenseur ")

def joueurALInterieur():
    return True

def monter(avecJoueur):
    print("monter")

def descendre(avecJoueur):
    print("descendre")

def atteindreEtage(y):
    return None

def verifierDemandes():
    print("verifierDemandes")

def etageAtteint(etage):
    print("etageAtteint")

def demandeSuivante(etage):
    print("demandeSuivante")

def ascenseurManuel():
    print("ascenseurManuel")

def ascenseurAutomatique():
    print("ascenseurAutomatique")

def detruireAscenseur():
    print("detruireAscenseur")
```

PERCER LES SECRETS DU CODE

Dans les premières lignes de ce programme, tu as établi la liste suivante :

```
demandes_enregistrees = [False, False, False, False]
```

Il s'agit d'une liste Python, comme celles que tu as utilisées dans les chapitres précédents. Tu lui as affecté quatre valeurs initiales, à savoir la valeur `False`. Souviens-toi que tu peux atteindre l'élément d'une liste en indiquant le numéro de sa position entre crochets, comme ceci :

```
demandes_enregistrees = [0]
```

Il est tout à fait possible d'utiliser des variables séparées, par exemple `etage1`, `etage2`, `etage3` et `etage4`. Et tu aurais très bien pu choisir de rédiger ton programme avec ces variables, mais cela t'aurait posé deux problèmes. Tout d'abord, cela l'aurait considérablement allongé, car tu aurais été obligé d'écrire plusieurs lignes de code pour vérifier les demandes éventuelles de chaque étage. En revanche, en utilisant une liste Python, tu peux passer en revue tous les étages d'un seul coup à l'aide d'une simple boucle, comme ceci :

```
for demande in demandes_enregistrees:
    print(str(request))
```

Enfin, imagine que tu souhaites ajouter quatre étages supplémentaires à ton ascenseur. En utilisant des variables séparées, il te faudrait apporter beaucoup de modifications et de développements à ton programme pour arriver à tes fins. En revanche, en utilisant une liste Python, tout ce qu'il te resterait à faire serait d'ajouter quatre valeurs `False` à la fin de la liste, et toutes les boucles `for` de ton programme s'ajusteraient automatiquement au nouveau nombre d'étages.

Utiliser des listes et des boucles `for` dans Python représente un gain de temps considérable et te permet de raccourcir et de modifier ton programme très simplement.

Écrire la boucle de jeu principale

En définissant toutes les fonctions du programme – une fonction pour chaque fonctionnalité – tu simplifies la boucle de jeu principale. Cela signifie également qu'après avoir fini d'écrire la structure du programme, tu peux le tester très rapidement, puis ajouter et tester de nouvelles fonctionnalités petit à petit.

La boucle de jeu principale ressemble à la plupart des programmes GPIO que tu as écrits dans les chapitres précédents. Elle doit être contenue dans le couple d'instructions `try/finally`. Puis elle doit appeler la fonction `GPIO.cleanup()` à la fin pour réinitialiser toutes les broches GPIO si tu appuies sur `Ctrl+F6` ou si tu mets fin au programme de manière abrupte. Pour la rédiger, tu vas suivre ces quelques instructions.

1. Le code principal se divise en deux parties. La première consiste à paramétrer tous les composants du dispositif et à construire l'ascenseur. Saisis les lignes suivantes à la fin de ton fichier `ascenseur.py` :

```
try:
    GPIO.setmode(GPIO.BCM)
    for p in ETAGE_GPIO:
        GPIO.setup(p, GPIO.IN)
    display.setup(GPIO, AFFICHEUR_GPIO, ON)

    creerAscenseur()
    tracerAscenseur(ascenseur_y, block.STONE.id)
```

2. La deuxième partie de ton code principal est réservée à la boucle de `jeu`, qui a pour fonction de vérifier constamment si l'ascenseur a été appelé avec les boutons et d'appeler la fonction `ascenseurManuel()`, qui s'occupera du reste. Enfin, n'oublie pas d'ajouter la fonction `GPIO.cleanup()` pour sécuriser toutes tes broches GPIO avant l'arrêt du programme. L'instruction `while True:` est déjà indentée d'un niveau, car elle fait partie du bloc `try:` :

```
    while True:
        time.sleep(0.5)
        verifierDemandes()
        ascenseurManuel()
finally:
    GPIO.cleanup()
```

Enregistre ton programme et exécute-le. Souviens-toi que sur Raspberry Pi, tu dois utiliser le module `sudo python lift.py` depuis une fenêtre LXTerminal.

Que se passe-t-il ? Eh bien, pas grand-chose à ce stade. Tu peux le constater sur la figure 10-4, qui te montre le résultat de ton programme une fois exécuté. Pourtant, tu viens de réaliser la plus grosse partie du travail. Et comme tu as écrit la plus grande partie de la structure du programme, il te sera désormais bien plus facile d'ajouter progressivement des fonctionnalités et de les tester une à une. Une fois les tests terminés, n'oublie pas de réinitialiser l'interpréteur de commandes ou d'appuyer sur `Ctrl+F6` dans la fenêtre de ton interpréteur, afin d'arrêter le programme. Sans cela, tu risques de rencontrer des problèmes dans les prochaines étapes.

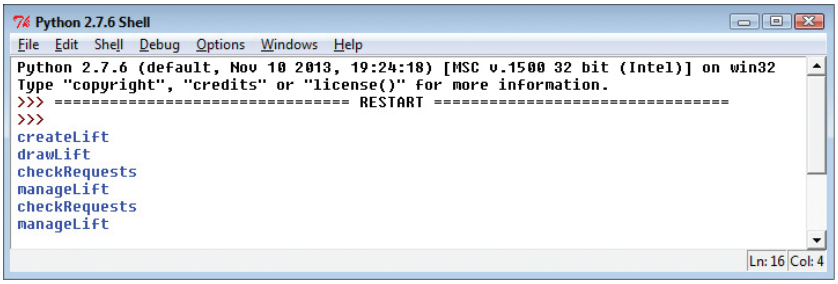


FIGURE 10-4 Le tout premier résultat de ton programme d'ascenseur

Mettre l'ascenseur en service

Même si ton ascenseur commence à prendre forme, il ne se passe toujours rien dans le monde de Minecraft. Alors pour l'instant, on ne peut pas vraiment parler d'un ascenseur Minecraft. Corrigeons cela tout de suite !

Créer l'ascenseur

Avant d'aller plus loin, tu dois d'abord construire ton ascenseur dans Minecraft. Pour l'heure, contente-toi de commencer simplement, en construisant une dalle en pierre. Comme tu vas créer l'ascenseur avec la fonction `dessinerAscenseur()`, il sera facile pour toi de modifier son apparence sans effectuer trop de changements dans ton programme. Voici comment tu vas t'y prendre.

1. Modifie ta fonction `creerAscenseur()` pour qu'elle ressemble à cela :

```
def creerAscenseur():
    for etage in range(NOMBRE_ETAGES):
        y = ASCENSEUR_Y + etage*HAUTEUR_ETAGE
        mc.setBlocks (ASCENSEUR_X-2, y, ASCENSEUR_Z-2,
                      ASCENSEUR_X+2, y+ HAUTEUR_ETAGE,
                      ASCENSEUR_Z+2, block.AIR.id)
        mc.setBlocks (ASCENSEUR__X+2, y, ASCENSEUR_Z-1,
                      ASCENSEUR_X+2, y, ASCENSEUR_Z+1,
                      block.DIAMOND_BLOCK.id)
```

   Cette fonction passe en revue tous les étages à l'aide d'une boucle et creuse une zone d'air rectangulaire entre chaque étage pour façonner ta cage d'ascenseur. Puis elle place une rangée de blocs de diamant à chaque étage. Pour appeler l'ascenseur, tu dois frapper l'un de ces blocs au niveau où se situe ton joueur.

2. Modifie la fonction `dessinerAscenseur()` pour qu'elle construise une dalle rectangulaire composée d'une série de blocs. L'identité du bloc que tu utilises lorsque tu dessines ton ascenseur est fournie par la valeur `blockid` ; elle te permet également d'« effacer » l'ascenseur en utilisant la même fonction avec des blocs d'air (`block.AIR.id`). Dessiner et effacer ton ascenseur est la manière la plus simple de le mettre en mouvement :

```
def dessinerAscenseur(y, blockid):
    mc.setBlocks(ASCENSEUR_X-1, y, ASCENSEUR_Z-1,
                 ASCENSEUR_X+1, y, ASCENSEUR_Z+1, blockid)
```

Enregistre ton programme et exécute-le. N'oublie pas que sur Raspberry Pi, tu dois utiliser `sudo python lift.py` depuis une fenêtre LXTerminal. Sur la figure 10-5, tu peux voir à quoi ressemble l'ascenseur dans Minecraft, avec la dalle en pierre qui sert à te transporter et les rangées de blocs de diamant situées à chaque étage desservi. Plus tard, tu pourras construire un bâtiment tout autour de cet ascenseur et les blocs de diamant te serviront à définir l'emplacement des étages.

FIGURE 10-5 L'ascenseur Minecraft, avec les boutons d'appel situés à chaque étage

Détecter le déclenchement des boutons d'appel électroniques

Dans le chapitre 5, tu as écrit un programme te permettant de détecter le déclenchement d'un bouton-poussoir. Détecter quatre boutons n'est pas singulièrement plus compliqué, mais comme il y en a plus d'un, ce sera plus pratique d'utiliser une boucle pour les examiner chacun à leur tour. Cela signifie également que si tu construis une structure plus grande, qui requiert par exemple l'installation de huit boutons, il te suffira de modifier la variable `ETAGE_GPIO` et le code fonctionnera exactement de la même manière.

1. Modifie la fonction `verifierDemandes()` pour qu'elle ressemble à cela :

```
def verifierDemandes():
    global demandes_enregistrees
    for etage in range(NOMBRE_ETAGES):
        if GPIO.input(ETAGE_GPIO[etage]) == False:
            demandes_enregistrees[] = True
    print(str(demandes_enregistrees))
```

Cette fonction passe chaque étage en revue à l'aide de la boucle `for` en obtenant leur numéro de broche GPIO grâce à la variable `ETAGE_GPIO`. Si elle lit la valeur `False` du dispositif GPIO (ce qui veut dire que le bouton est connecté à la bande de 0 volt et, donc, qu'il est actionné), elle attribue la valeur `True` à l'entrée `demandes_enregistrees`. Cela signifie qu'un appel a été effectué depuis l'un des étages et qu'il attend une réponse.

2. Enregistre ton programme et exécute-le. N'oublie pas que sur Raspberry Pi, tu dois utiliser `sudo python lift.py` depuis une fenêtre LXTerminal. Appuie sur les boutons de la plaque d'essais lorsque ton programme s'exécute et observe bien ce qu'il se passe !

   Sur la figure 10-6, tu peux voir ce qu'il se passe lorsque tu exécutes ce programme. Chaque demi-seconde, l'instruction `print()` de la fonction `verifierDemandes()` affiche dans l'interpréteur de commandes Python les étages qui ont fait l'objet d'un appel et qui attendent d'être desservis par l'ascenseur. Comme ton programme définit les valeurs de la liste `demandes_enregistrees` à chaque fois que tu appuies sur un bouton, il conserve ces demandes en mémoire.

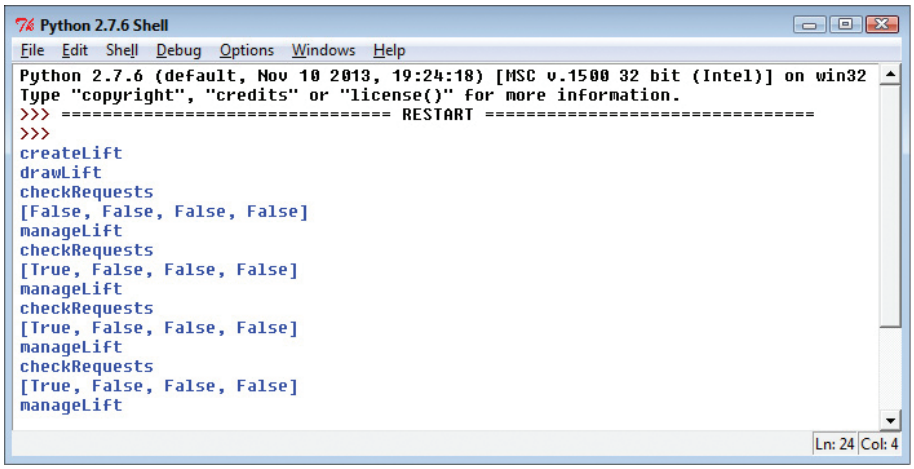


FIGURE 10-6 L'interpréteur de commandes Python montrant l'évolution de la variable demandes_enregistrees à mesure que les boutons sont actionnés.

Lorsque j'ai écrit ce programme, l'idée que l'ordre des `demandes_enregistrees` et celui des boutons soient inversés et créent la confusion ne me plaisait guère. J'ai toutefois préféré ne pas remédier à ce défaut, car la fonction `print` n'est qu'une méthode temporaire de test et ne présente un intérêt que pour toi, en tant que programmeur.

Lorsque tu auras terminé le programme de l'ascenseur, le message de ce test n'aura plus aucune importance, donc il n'est pas nécessaire de perdre ton temps à effectuer des tests `print` de débogage trop sophistiqués. Parfois, les messages de débogage n'ont réellement de sens que pour la personne qui a écrit le programme, ce qui est tout à fait normal !

Utiliser un mode de service pour activer l'ascenseur manuellement

La prochaine étape du développement de ton programme consiste à trouver un moyen de faire monter et descendre ton ascenseur manuellement, pour savoir s'il se déplace correctement.

Lorsque tu développes un long programme en plusieurs étapes comme celui-ci, il faut parfois écrire des blocs de code temporaires dont tu te débarrasseras ensuite. Les développeurs de logiciels appellent ce procédé l'échafaudage (ou le *scaffolding*), parce que cela ressemble un peu aux échafaudages que les ouvriers montent autour d'un bâtiment lors de sa construction. Ils permettent de maintenir la structure pendant les travaux de construction et sont ensuite démontés lorsque les travaux sont finis. Tu vas utiliser cette technique d'échafaudage ici, pour écrire un mode de service qui te permettra de faire fonctionner ton ascenseur manuellement jusqu'à ce que tu termines te rédiger le reste du programme de l'ascenseur.

1. La fonction `monter()` réalise deux choses : elle monte la dalle de l'ascenseur d'un bloc et fait de même pour le joueur. Modifie ta fonction `monter()` en saisissant le code suivant :

```
def monter(avecJoueur):
    global ascenseur_y
    if avecJoueur:
        mc.player.setPos(ASCENSEUR_X, ascenseur_y+2,
                         ASCENSEUR_Z)
    dessinerAscenseur(ascenseur_y+1, block.STONE.id)
    dessinerAscenseur(ascenseur_y, block.AIR.id)
    if avecJoueur:
        mc.player.setPos(ASCENSEUR_X, ascenseur_y+2,
                         ASCENSEUR_Z)
    ascenseur_y = ascenseur_y+1
```

2. Maintenant, modifie la fonction `descendre()` de la sorte :

```
def descendre(avecJoueur):
    global ascenseur_y
    dessinerAscenseur(ascenseur_y-1, block.STONE.id)
    dessinerAscenseur(ascenseur_y, block.AIR.id)
    if avecJoueur:
        mc.player.setPos(ASCENSEUR_X, ascenseur_y+1,
                         ASCENSEUR_Z)
    ascenseur_y = ascenseur_y-1
```

La fonction `descendre()` est identique à la fonction `monter()`. Remarque que dans ces deux fonctions, `avecJoueur` est utilisé pour déplacer le joueur en même temps que l'ascenseur. Cela te sera utile par la suite lorsque tu voudras déplacer l'ascenseur avec ton joueur à l'intérieur ou à l'extérieur.

3. Pour créer un mode de service au sein de la fonction `ascenseurManuel()`, saisis le code suivant :

```
def ascenseurManuel():
    global demandes_enregistrees
    if demandes_enregistrees[0]:
        monter(True)
        demandes_enregistrees[0]:
    elif demandes_enregistrees[1]:
        descendre(true)
        demandes_enregistrees[1] = False
```

Ce bloc permet simplement d'utiliser les deux premiers boutons pour vérifier que ton ascenseur fonctionne correctement. Si une demande est enregistrée à l'aide du premier bouton, l'ascenseur devrait monter d'un bloc. En revanche, si tu appelles l'ascenseur avec le deuxième bouton, l'ascenseur devrait descendre d'un bloc.

Enregistre ton programme et exécute-le. N'oublie pas d'utiliser `sudo python lift.py` depuis une fenêtre LXTerminal si tu utilises un Raspberry Pi. Appuie sur les deux premiers boutons pour voir si ton ascenseur monte et descend comme prévu.

Que se passe-t-il si tu continues d'appuyer sur le deuxième bouton une fois que tu as atteint le seuil de la cage d'ascenseur ? Et que se passe-t-il lorsque tu continues d'appuyer sur le premier bouton une fois que tu as atteint la hauteur maximale ? Comme tu peux le voir, le mode de service temporaire pose un problème. Penses-tu qu'il faut le résoudre, même si ce mode n'est qu'un test temporaire ?

Afficher le numéro des étages

Ton ascenseur est presque terminé et tu vas bientôt l'automatiser. Mais avant cela, tu dois ajouter quelques notions de mathématiques à ton programme, pour qu'il puisse calculer l'étage auquel ton ascenseur se trouve. Cela servira notamment à mettre à jour l'afficheur 7 segments et à afficher le numéro de l'étage à mesure que ton ascenseur monte et descend. Comme tu le sais, tu ne peux pas vraiment voir la cage d'ascenseur dans ton cas, car tu n'as pas construit d'immeuble autour. Mais si tu construis ton ascenseur au beau milieu d'une montagne ou le long d'une colline, tu la verras tout de suite.

1. Modifie la fonction `atteindreEtage()` de la sorte :

```
def atteindreEtage(y):
    reste = (y-ASCENSEUR_Y) % HAUTEUR_ETAGE
    if reste == 0:
        return (y-ASCENSEUR_Y) / HAUTEUR_ETAGE
    return None
```

   La variable `y` correspond à la position y de l'ascenseur dans Minecraft. Elle utilise la position de l'ascenseur dans le jeu et les numéros des étages pour calculer l'étage auquel il se trouve. S'il se trouve entre deux étages, elle renvoie la valeur `None`. Pour en savoir plus sur le calcul qu'effectue cette fonction, lis l'encadré « Percer les secrets du code ».

2. À la fin de la fonction `ascenseurManuel()`, ajoute les lignes en gras du bloc suivant. Fais bien attention à l'indentation dans ces nouvelles lignes ; elles sont indentées d'un cadratin, car elles font partie du corps de la fonction `ascenseurManuel()` :

```
    elif demandes_enregistrees[1]:
        descendre(True)
        demandes_enregistrees[1] = False
    etage = atteindreEtage(ascenseur_y)
    if etage != None:
        display.write(str(etage))
```

Enregistre ton programme et exécute-le. Souviens-toi que sur Raspberry Pi, tu dois utiliser `sudo python lift.py` depuis une fenêtre LXTerminal. Appuie sur les deux premiers boutons pour faire monter et descendre ton ascenseur le long de la cage et assure-toi que les numéros qui s'affichent correspondent bien à ceux des étages.

PERCER LES SECRETS DU CODE

Tu viens d'effectuer un nouveau tour de magie dans la fonction `atteindreEtage()`, alors étudions-la d'un peu plus près. Voici à quoi ressemble ta fonction `atteindreEtage()` :

```
def atteindreEtage(y):
    reste = (y-ASCENSEUR_Y) % HAUTEUR_ETAGE
    if reste == 0:
        return (y-ASCENSEUR_Y) / HAUTEUR_ETAGE
    return None
```

Lorsque tu as construit ta cage d'ascenseur et que tu as placé des blocs de diamant à chaque étage, tu les as placés en fonction de la position y qu'utilisait la boucle pour façonner la cage d'ascenseur. Ces nouvelles lignes de code font en quelque sorte le calcul inverse, c'est-à-dire qu'elles calculent le numéro de l'étage à partir de la position `y` de ton ascenseur.

Les étages sont numérotés de 0 à 3 et chacun d'entre eux mesure l'équivalent de la variable `HAUTEUR_ETAGE`. Donc, si tu divises la position `y` par la variable `HAUTEUR_ETAGE` et que tu tombes sur un nombre strictement positif (supérieur à 0), cela signifie que ton ascenseur a atteint un étage. Si après la division tu ne tombes pas sur un nombre strictement positif, cela veut dire que ton ascenseur se trouve quelque part entre deux étages.

Pour le savoir, tu dois utiliser un reste. Le langage Python dispose d'un symbole spécial, l'opérateur `%`, appelé *modulo*, qui correspond en quelque sorte au reste. Si ta position `y` équivaut à 5 et que chaque étage mesure 4 blocs de haut, ton calcul est le suivant : 5/4 = 1 et il reste 1 (autrement dit, le modulus est égal à 1). Si ta position `y` équivaut à 4 et que chaque étage mesure 4 blocs de haut, tu obtiendras : 4/4 = 1 reste 0 (le modulus est égal à 0). Lorsqu'il ne reste rien après la division, cela veut dire que la position `y` de l'ascenseur est alignée sur celle de l'un des étages. Pour en savoir plus sur l'opérateur modulo, tu peux te rendre sur le site docs de Python : `https://docs.python.org/2/reference/expressions.html`.

Si le reste n'est pas nul (c'est-à-dire que l'ascenseur se trouve entre deux étages), la fonction `atteindreEtage()` renvoie la valeur spéciale `None`. Cette dernière est utilisée ailleurs dans le programme pour détecter que l'ascenseur n'a pas encore atteint un étage.

Automatiser l'ascenseur

Tu as maintenant fini d'écrire les différentes parties de ton programme et elles fonctionnent toutes correctement. La dernière étape consiste à assembler tous ces blocs et à les saupoudrer d'une pincée d'intelligence pour automatiser ton ascenseur et lui faire atteindre d'autres étages une fois qu'il s'est arrêté là où tu l'as appelé. Une fois cette étape achevée, ton ascenseur sera entièrement fonctionnel et tu pourras l'installer dans n'importe quel bâtiment que tu auras construit dans le monde de Minecraft, et ce quelle que soit sa taille !

Les fonctions marche et arrêt

Tu as déjà ajouté le code te permettant de vérifier si ton ascenseur se trouve à un étage, car les numéros des étages s'affichent désormais sur ton afficheur 7 segments dès qu'il les a atteints. Mais un vrai ascenseur s'arrête toujours au niveau d'où on l'a appelé une fois qu'il l'a atteint. Tu vas reproduire ce comportement en suivant les instructions ci-après.

1. Modifie ta fonction `etageAtteint()` en saisissant les lignes suivantes :

```
def etageAtteint(etage):
    global etat_ascenseur, demandes_effectuees
```

```
    if demandes_effectuees[etage]:
        demandes_enregistrees[etage] = False
        etat_ascenseur = ARRET
        display.write(str(etage))
        time.sleep(2)
```

Ces nouvelles lignes de code vérifient la liste des `demandes_effectuees` pour savoir à quel étage se trouve l'ascenseur. Si un étage fait l'objet d'un appel, l'ascenseur s'y arrête. Lorsque l'ascenseur s'arrête, le programme affiche le numéro de l'étage sur l'afficheur 7 segments et attend deux secondes pour laisser le temps au joueur d'entrer ou de sortir, avant qu'il ne reparte pour une autre tournée de dessertes !

2. Une fois que l'ascenseur s'est arrêté à un étage et qu'il est prêt à repartir, il doit choisir le prochain étage à atteindre. Pour ce faire, remplace la fonction `demandeSuivante()` par le code suivant :

```
def demandeSuivante():
    global etat_ascenseur
    for f in range(NOMBRE_ETAGES):
        if demandes_enregistrees[f]:
            if f > etage:
                etat_ascenseur = ASCENSION
            else:
                etat_ascenseur = DESCENTE
            return
```

Pour choisir le prochain étage à atteindre, le code utilise une méthode toute simple : il scanne toutes les demandes du plus bas niveau jusqu'au plus élevé jusqu'à ce qu'il en trouve une, ce qui veut dire que l'étage 0 a la priorité sur tous les autres, comme pour les vrais ascenseurs.

Comme tu peux le voir, tu as utilisé l'instruction `return` à la fin de ta fonction. Elle permet d'arrêter la recherche dès qu'une demande a été trouvée, puis revient au début de la fonction sans mettre fin à la boucle `for`.

Achever le panneau de contrôle de l'ascenseur

Ton ascenseur est bientôt terminé ! Les fonctions que tu viens d'écrire renferment déjà toute l'intelligence dont ton ascenseur a besoin. Maintenant, il ne te reste plus qu'à assembler tous les morceaux de ton programme et à le tester !

1. Remplace la fonction `ascenseurAutomatique()` par les lignes suivantes :

```
def ascenseurAutomatique():
    avecJoueur = joueurALInterieur()
    etage = atteindreEtage(ascenseur_y)
```

```
    if etage != None:
        etageAtteint(etage)
    if etat_ascenseur == ASCENSION:
        display.write("monte")
        monter(avecJoueur)
    elif etat_ascenseur == DESCENTE:
        display.write("descend")
        descendre(avecJoueur)
    else:
        demandeSuivante(etage)
```

Cette portion de code permet de détecter si l'ascenseur monte, descend ou s'arrête à un étage. La magie qui s'opère à chacun de ces états a été programmée dans les fonctions que tu as rédigées un peu plus haut.

2. Modifie la boucle de jeu principale en supprimant la ligne `ascenseurManuel()` et en la remplaçant par la fonction `ascenseurAutomatique()`.

   Enregistre ton programme et exécute-le. N'oublie pas d'utiliser `sudo python lift.py` dans une fenêtre LXTerminal si tu utilises un Raspberry Pi.

Teste tous les étages à l'aide des boutons. Tu dois tester le programme de ton ascenseur de manière très méticuleuse en appuyant sur tous les boutons pour que ce dernier s'arrête à tous les étages. Lors de ton test, tu vérifieras également les numéros et l'ordre dans lequel ton ascenseur se déplace.

DÉFI

Amuse-toi à construire une tour tout autour de la cage d'ascenseur, avec plusieurs salles à chaque étage. Caches-y des trésors impossibles à trouver et demande à tes amis de les déceler en explorant tous les niveaux de la tour.

Faire fonctionner l'ascenseur lorsque le joueur se trouve à l'extérieur

Pour finaliser ton ascenseur, il ne te reste plus qu'à le faire fonctionner lorsque ton joueur n'est pas dedans. Tu dois donc compléter la fonction `joueurALInterieur()` de façon à géorepérer la cage d'ascenseur et à détecter lorsque les rangées de blocs de diamant placées aux étages ont été frappées. Ainsi, l'ascenseur pourra obéir aux demandes de ton joueur et le transporter où il le souhaite.

Détecter lorsque le joueur se trouve dans l'ascenseur

Ton programme doit pouvoir détecter si ton joueur se trouve à l'intérieur ou à l'extérieur de l'ascenseur, afin de savoir s'il doit `monter()` ou `descendre()` avec ton joueur ou s'il doit désactiver les boutons de ton panneau de commandes. En effet, tu vas le désactiver lorsque ton joueur se trouvera à l'extérieur de l'ascenseur ; tu ne pourras simplement plus actionner les boutons de la plaque !

Dans le chapitre 2, tu as appris la technique du géorepérage. Le moyen le plus simple de détecter si ton joueur se trouve à l'intérieur de l'ascenseur, c'est de géorepérer toute la cage d'ascenseur. Ton programme dispose déjà d'une fonction d'apparat `joueurALInterieur()`, donc tout ce qu'il te reste à faire est de développer cette fonction.

1. Remplace la fonction `joueurALInterieur()` par le code suivant :

```
def joueurALInterieur():
    pos = mc.player.getPos()
    if pos.x >= ASCENSEUR_X-1 and pos.x <= ASCENSEUR_X+1 ↩
    and pos.z >= ASCENSEUR_Z-1 and pos.z <= ASCENSEUR_Z+1:
        return True
    return False
```

   Les coordonnées de la cage d'ascenseur sont utilisées pour son géorepérage. La fonction renvoie la valeur `True` si le joueur se trouve à l'intérieur et `False` dans le cas contraire.

2. Modifie la fonction `verifierDemandes()` de façon à activer les boutons du dispositif lorsque ton joueur se trouve à l'intérieur de l'ascenseur. Attention à l'indentation dans le corps de la fonction ; tu dois indenter la boucle `for` existante d'un cadratin supplémentaire pour qu'elle fasse partie de l'instruction `if` :

```
def verifierDemandes():
    global demandes_enregistrees

    if joueurALInterieur():
        for etage in range(NOMBRE_ETAGES):
            if GPIO.input(ETAGE_GPIO[etage]) == False:
demandes_enregistrees[etage] = True

    print(str(demandes_enregistrees))
```

3. Enregistre ton programme et exécute-le. N'oublie pas d'utiliser `sudo python lift.py` depuis une fenêtre LXTerminal si tu travailles sur un Raspberry Pi.

Détecter lorsque ton joueur frappe un bloc pour appeler l'ascenseur

Désormais, dès que ton joueur se trouve en dehors de l'ascenseur, les boutons du panneau de commande sont désactivés. C'est assez logique, car s'il s'agissait d'un vrai ascenseur, tu ne pourrais tout simplement pas accéder à son panneau de commandes qui se trouve à l'intérieur. Cela étant, il faut maintenant que tu trouves un moyen de l'appeler depuis l'extérieur.

Dans le chapitre 4, tu as appris à utiliser la fonction `pollBlockHits()` pour détecter lorsque ton joueur frappe un bloc. Tu peux utiliser exactement la même technique ici pour détecter lorsque ton joueur frappe l'un des blocs de diamant situés à l'étage où il se trouve, afin de faire venir l'ascenseur.

1. Insère ces nouvelles lignes de code dans la fonction `verifierDemandes()`, juste en dessous de la ligne `global demandes_enregistrees` :

```
def verifierDemandes()
    global demandes_enregistrees

    for e in mc.events.pollBlockHits():
        pos = e.pos
        etage = atteindreEtage(pos.y)
        if etage != None:
            demandes_enregistrees[etage] = True
```

   Le reste de la fonction ne doit pas être modifié, car ton joueur a besoin d'utiliser le panneau de commandes s'il pénètre à l'intérieur de l'ascenseur ! Ces nouvelles lignes de code ont exactement la même fonction que celles du chapitre 4 : elles récupèrent les coordonnées des blocs qui ont été frappés. Heureusement, dans l'une des étapes précédentes, tu as développé la fonction `atteindreEtage()`, qui te permet de transformer une coordonnée y en numéro d'étage. Ton programme peut donc facilement détecter les blocs qui ont été frappés et envoyer l'ascenseur à l'étage concerné. Enfin, pour l'appeler, il te suffit simplement d'enregistrer l'appel dans la liste `demandes_enregistrees[]`.

2. Enregistre ton programme et exécute-le. N'oublie pas d'utiliser `sudo python lift.py` depuis une fenêtre LXTerminal si tu disposes d'un Raspberry Pi.

Vérifie que tout fonctionne correctement. Si tu te rends à l'un des étages et que tu frappes avec ton épée l'un des blocs de diamants situés en face de la cage d'ascenseur, celui-ci ne devrait pas tarder à arriver pour t'amener là où tu le souhaites. Monte sur la dalle et utilise ton panneau de commandes pour te rendre à l'étage de ton choix. Tu es désormais libre de monter et descendre comme bon te semble dans le monde de Minecraft !

Sur la figure 10-7, tu peux voir l'ascenseur s'élever vers ton joueur pour l'accueillir, car ce dernier vient juste de l'appeler en frappant les blocs de diamant avec son épée.

La fonction `pollBlockHits()` que tu as saisie ne vérifie que la coordonnée `y` pour détecter si le joueur a frappé l'un des blocs de diamant. Que se passe-t-il si ton joueur frappe un autre diamant à la même hauteur dans Minecraft ? Fais-en l'essai et observe ce qu'il se passe. Peux-tu remédier à ce problème en modifiant le code de ta fonction `verifierDemandes()` ?

FIGURE 10-7 L'ascenseur monte vers le joueur.

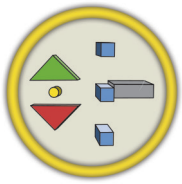
DÉFI

Écris une fonction `detruireAscenseur()` pour détruire la cage d'ascenseur et l'ascenseur lui-même une fois ton programme terminé. Où dois-tu la placer dans ton programme pour qu'elle fonctionne correctement ? Teste-la en appuyant sur les touches `Ctrl+F5` dans ton interpréteur de commandes Python (ou dans une fenêtre LXTerminal de ton Raspberry Pi). Vérifie qu'il ne reste plus aucune trace de ton ascenseur après avoir refermé ton programme.

Explorer d'autres pistes avec l'ascenseur Minecraft

Dans ce chapitre, tu as élaboré un fantastique logiciel et confectionné un superbe panneau de commandes pour construire un ascenseur entièrement fonctionnel, exactement comme ceux que tu peux voir dans les vrais immeubles. Ce faisant, tu as fait appel à toutes les compétences que tu as acquises dans les chapitres du livre, comme détecter la position de ton joueur, géorepérer la cage d'ascenseur, déplacer ton joueur d'un endroit à un autre, construire des blocs automatiquement, détecter lorsqu'un bloc a été frappé, interagir avec un circuit électronique, utiliser des listes Python, développer et tester un programme pas à pas, et bien d'autres choses encore !

- Cet ascenseur est on ne peut plus simple : il s'agit tout bonnement d'une dalle de pierre qui t'emmène là où tu le souhaites ! Utilise le module `MinecraftStuff` que Martin a conçu pour toi, comme tu as appris à le faire dans les chapitres 7, 8 et 9, puis utilise une forme `MinecraftShape` pour construire un ascenseur plus réaliste. Tu pourrais par exemple utiliser des blocs de verre pour que ton joueur puisse voir le monde de Minecraft depuis l'ascenseur. Appelle ensuite la fonction `setBlocks()` pour le doter de portes coulissantes s'ouvrant et se refermant à chaque étage. Tu pourrais même t'amuser à télécharger le module `pygame` et utiliser le son `pygame.mixer.Sound("swoosh.wav")` pour simuler le son des portes ! Pour en savoir plus sur `pygame`, consulte l'annexe A du livre, intitulée « Pour aller plus loin ».
- Tu trouveras tout ce que tu dois savoir sur la conception d'ascenseurs en lisant cet article complet sur Wikipédia : `https://fr.wikipedia.org/wiki/Ascenseur`. Étudie leurs différentes caractéristiques et essaye d'ajouter de nouvelles fonctionnalités à ton engin, en donnant par exemple la priorité à des étages spécifiques à certains moments de la journée.



Niveau terminé : tu as fini la construction de ton ascenseur Minecraft ! Tu peux maintenant te rendre au sommet d'un gratte-ciel en un clin d'œil !

Annexe B
Tableau de références

Tableau 1 – Les variables d'entrée et de sortie

Laisser des rappels grâce aux commentaires	Utiliser les constantes
`# commentaire sur la ligne entière` `print("bonjour") # commentaire en ↩ fin de ligne`	`DIMENSIONS = 100 # nom en majuscules`
Afficher du contenu à l'écran	**Lire du texte saisi au clavier (Python 2)**

`print(10) # afficher un nombre`
`print("bonjour") # afficher une ↩ chaîne de caractères`
`print(a) # afficher une variable`
	`nom = raw_input("quel est ton nom ?")`
	`age = int(rawinput("quel âge as-tu ?"))`
	`ageDansUnAn = age + 1`

Affecter des valeurs à des variables	**Utiliser des variables booléennes**

`a = 10 # un nombre entier`
`b = 20.2 # un nombre décimal`
`message = "bonjour" # une chaîne ↩ de caractères`
`fini = True # une variable booléenne`
`resultat_eleve = None # aucune valeur`
	`anotherGo = True`
	`while anotherGo:`
	`    choix = int(raw_input ("choix ?"))`
	`    if choix == 8:`
	`        anotherGo = False`

Convertir des chaînes de caractères en nombres (entiers)	**Convertir des nombres en chaînes de caractères**
`dimensions = int(dimensions)`	`print("ton age :" + str(age))`
Calculer avec des nombres variables	**Calculer des valeurs absolues et des différences**

`b = a + 1 # addition`
`c = a - 1 # soustraction`
`d = a * 2 # multiplication`
`e = a / 2 # division`
`f = a % 2 # reste après division`
	`absolu = abs(-1) # le nombre ↩ absolu équivaut à 1`
	`difference = abs(x1-x2)`
	`minimal = min(x1, x2)`
	`maximal = max(x1, x2)`

Tableau 2 – Les boucles Python

Écrire une boucle	Utiliser des variables de comptage

`a = 0`
`while a<100:`
`    print(a)`
`a = a + 1`
	`for a in range(10):`
	`    print(a) # de 0 à 9`
	`        for a in range(5, 50, 5):`
	`print(a) # de 5 à 45 par séries de 5`

Passer en revue tous les caractères d'une chaîne	**Séparer des chaînes de caractères par des virgules**

`nom = "Charlotte"`
`for caractere in nom:`
`    print(caractere)`
	`ligne = "un,deux,trois"`
	`donnee = line.split(",")`
	`for d in donnees:`
	`    print(d)`

Tableau 3 – Les conditions dans Python

Utiliser des instructions if	Utiliser des instructions if/else

`if a == 42:`
`    print("la dernière réponse")`
	`if a == 5:`
	`    print("correct")`
	`else:`
	`    print("faux")`

Utiliser elif/else pour ultiples	**Vérifier plusieurs conditions avec *and* et *or***

`if a==1:`
`    print("option A")`
`elif a==2:`
`    print("option B")`
`else:`
`    print("tout le reste")`
	`if choix>0 and choix<=8:`
	`    print("inclus")`
	`if choix<1 or choix>8:`
	`    print("exclus")`

Vérifier si des valeurs sont différentes ou égales avec *if*	**Vérifier si des valeurs sont inférieures avec *if***

`if a!=1:`
`    print("différent")`
`if a==1:`
`    print("égal")`
	`if a<1:`
	`    print("inférieur a")`
	`if a<=1:`
	`    print("inférieur a ou égal")`

Vérifier des valeurs opposées avec *not*	**Vérifier si des valeurs sont supérieures avec *if***

`jeu = True`
`if not jeu:`
`    print("terminé")`
	`if a>1:`
	`    print("supérieur à")`
	`if a>=1:`
	`    print ("supérieur à ou égal")`

Tableau 4 – Les fonctions Python

Définir et appeler des fonctions	Passer valeurs en paramètre dans des fonctions

`def monnom():`
`  print("mon nom est Laura")`
`monnom()`
	`def bonjour(nom):`
	`  print("bonjour " + nom)`
	`bonjour("André")`
	`bonjour("Géraldine")`

Renvoyer des valeurs à partir de fonctions	**Écrire dans des variables globales à l'intérieur de fonctions**

`def minimal(a, b):`
`  if a<b:`
`  return a`
`  return b`
`print(minimal(10, 20))`
	`tresor_x = 0`
	`def construction():`
	`  global tresor_x`
	`  tresor_x = 10`

Tableau 5 – Les listes Python

Créer des listes	Ajouter un objet à la fin d'une liste
`a = [] # une liste vide` `a = [1, 2, 3]# une liste commencée`	`a.append("bonjour")`
Afficher le contenu d'une liste	**Connaître la taille d'une liste**
`print(a)`	`print(len(a))`
Accéder aux objets d'une liste grâce à leur index	**Accéder au dernier objet d'une liste**
`print(a[0]) # 0=premier objet, 1=deuxième`	`print(a[-1])`
Supprimer le dernier objet d'une liste	**Retourner tous les objets d'une liste**

`mot = a.pop() # supprime le dernier objet`
`print(mot) # l'objet vient d'être supprimé`
	`for nom in ["David","Gail",`
	`  "Janet"]:`
	`  print(nom)`

Tableau 6 – D'autres modules Python	
Ajouter un petit intervalle de temps	**Produire des nombres aléatoires**
`import time` `time.sleep(1) # attend 1 seconde`	`import random` `print(random.randint(1,100))`
Obtenir la date et l'heure actuelles	**Utiliser des fonctions mathématiques**

`import datetime`
`dateActuelle = datetime.datetime.now()`
`import time`
`tempsActuel = time.time()`
	`import math`
	`radians = math.radians(angle)`
	`sin = math.sin(radians)`
	`cos = math.cos(radians)`
	`racineCarree = math.sqrt(nombre)`

Tableau 7 – Traitement de fichiers	
Lire des lignes dans un fichier	**Écrire des lignes dans un fichier**

`f = open("donnees.txt", "r")`
`astuces = f.readlines()`
`f.close()`
`for t in astuces:`
`    print(t)`
	`f = open("scores.txt", "w")`
	`f.write("Victoria:26000\n")`
	`f.write("Amanda:10000\n")`
	`f.write("Ria:32768\n")`
	`f.close()`

Obtenir une liste de fichiers analogues nomsdefichiers	**Enlever des espaces blancs en trop dans des chaînes de caractères**

`import glob`
`noms = glob.glob("*.csv")`
`for n in noms:`
`    print(n)`
	`a = "\n\n salut \n\n"`
	`a = a.strip()`
	`print(a)`

Tableau 8 – Brancher un dispositif électronique

Configurer les broches GPIO	Interpréter et contrôler des broches GPIO

`import RPi.GPIO as GPIO # RaspberryPi`
`import anyio.GPIO as GPIO # Arduino`
`GPIO.setmode(GPIO.BCM)`
`GPIO.setup(5, GPIO.OUT) # sortie`
`GPIO.setup(6, GPIO.IN) # entrée`
	`GPIO.output(5, True) # activée ↩ (3,3V)`
	`GPIO.output(5, False) # désactivée ↩ (0V)`
	`if GPIO.input(6) == False:`
	`  print("enfoncé")`

Remettre les broches GPIO en état après utilisation	**Utiliser le module de l'afficheur 7 segments**

`try:`
`  fais_quelquechose() # saisis du code ici`
`finally:`
`  GPIO.cleanup()`
	`import anyio.seg7 as display`
	`BROCHES_LED = [10,22,25,8,7,9,11, 15]`
	`ON = False # Anode commune`
	`ON = True # Cathode commune`
	`display.setup(GPIO, BROCHES_LED, ON)`

Afficher des caractères sur l'afficheur 7 segments	**Autres fonctions de l'afficheur 7 segments**

`display.write("3")`
`display.write("A")`
`display.write("up")`
	`display.setdp(True) # séparateur ↩ décimal activé`
	`display.setdp(False) # séparateur ↩ décimal désactivé`
	`display.clear()`
	`display.pattern([1,1,0,1,1,0,1,0])`

Tableau 9 – L'API de Minecraft

Ce sont les principales fonctions Python de l'API de Minecraft. Pour une liste plus complète, rends-toi sur : www.stuffaboutcode.com/p/minecraft-api-reference.html.

Importer l'API de Minecraft

```
import mcpi.minecraft as minecraft
```

Importer et utiliser les constantes des noms de blocs

```
import mcpi.block as block

b = block.DIRT.id
```

Se connecter à Minecraft

```
mc = minecraft.Minecraft.create()
```

Afficher un message dans le chat de Minecraft

```
mc.postToChat("Bonjour Minecraft")
```

Obtenir la position du joueur

```
# quelles sont les coordonnées
# du bloc sur lequel se trouve
# le joueur ?
pos = mc.player.getTilePos()
x = pos.x
y = pos.y
z = pos.z
```

Définir la position du joueur

```
x = 5
y = 3
z = 7
mc.player.setTilePos(x, y, z)
```

Obtenir la position précise du joueur

```
# obtenir la position précise
# du joueur
# dans le monde (par exemple :
# x=2.33)
pos = mc.player.getPos()
x = pos.x
y = pos.y
z = pos.z
```

Définir la position précise du joueur

```
# définir la position précise
# du joueur
x = 2.33
y = 3.95
z = 1.23
mc.setPos(x, y, z)
```

Connaître le niveau du sol

```
# coordonnée y du premier bloc
# autre qu'un bloc d'AIR
hauteur = mc.getHeight(5, 10)
```

Obtenir l'identité d'un bloc à un endroit précis

```
# numéro d'identité du bloc
# (par exemple : block.DIRT.id)
idBloc = mc.getBlock(10, 5, 2)
```

Définir/modifier l'identité d'un bloc à un endroit donné

```
mc.setBlock(5, 3, 2, block.DIRT.id)
```

Définir/modifier l'identité de plusieurs blocs à la fois

```
mc.setBlocks(0,0,0, 5,5,5,
             block.AIR.id)
```

Retrouver les blocs qui ont été frappés

```
# quels blocs ont été frappés
# depuis la dernière fois ?
evenements = mc.events.pollBlockHits()
for e in evenements:
    pos = e.pos
    print(pos.x)
```

Effacer toutes les frappes de bloc

```
# effacer(ignorer) les dernières
# frappes
mc.events.clearAll()
```

Tableau 10 – Types des blocs de l'API de Minecraft

Ce tableau répertorie certains des types de blocs disponibles dans Minecraft. Les valeurs de données sont indiquées lorsque cela s'avère utile et les colonnes Pi et PC/Mac indiquent si le bloc peut être utilisé sur ces systèmes. Pour obtenir une liste complète des numéros d'identité des blocs et de leurs valeurs de données, tu peux te rendre sur le site : www.stuffaboutcode.com/p/minecraft-api-reference.html.

ID	Constante	Valeur de donnée	Sous-catégorie	Pi	PC/MAC
0	AIR	-	-	O	O
1	ROCHE	-	-	O	O
2	HERBE	-	-	O	O
3	POUSSIERE	-	-	O	O
4	ROCHE	-	-	O	O
5	PLANCHES	0	Chêne	O	O
		1	Épicéa	N	O
		2	Bouleau	N	O
		3	Acajou	N	O
7	BEDROCK	-	-	O	O
8	EAU	-	-	O	O
9	EAU_STAGNANTE	0	Profonde	O	O
		7	Peu profonde	O	O
10	LAVE	-	-	O	O
11	LAVE_STAGNANTE	0	Profonde	O	O
		7	Peu profonde	O	O
12	SABLE	-	-	O	O
13	GRAVIER	-	-	O	O
14	MINERAI_D_OR	-	-	O	O
15	MINERAI_DE_FER	-	-	O	O
16	MINERAI_DE_CHARBON	-	-	O	O
17	BOIS	0	Chêne (haut/bas)	O	O
		1	Épicéa (haut/bas)	O	O
		2	Bouleau (haut/bas)	O	O
		3	Acajou (haut/bas)	N	O
		4	Chêne (est/ouest)	N	O
		5	Épicéa (est/ouest)	N	O
		6	Bouleau (est/ouest)	N	O
		7	Acajou (est/ouest)	N	O
		8	Chêne (nord/sud)	N	O
		9	Épicéa (nord/sud)	N	O

Tableau 10 *suite*

ID	Constante	Valeur de donnée	Sous-catégorie	Pi	PC/ MAC
		10	Bouleau (nord/sud)	N	O
		11	Épicéa (nord/sud)	N	O
		12	Chêne (écorce)	N	O
		13	Épicéa (écorce)	N	O
		14	Bouleau (écorce)	N	O
		15	Acajou (écorce)	N	O
18	FEUILLES	1	Feuilles de chêne	O	O
		2	Feuilles d'épicéa	O	O
		3	Feuilles de bouleau	O	O
20	VERRE	-	-	O	O
24	GRES	0	Grès	O	O
		1	Grès sculpté	O	O
		2	Grès poli	O	O
35	LAINE	0	Blanc	O	O
		1	Orange	O	O
		2	Magenta	O	O
		3	Bleu clair	O	O
		4	Jaune	O	O
		5	Vert clair	O	O
		6	Rose	O	O
		7	Gris	O	O
		8	Gris clair	O	O
		9	Cyan	O	O
		10	Violet	O	O
		11	Bleu	O	O
		12	Marron	O	O
		13	Vert	O	O
		14	Rouge	O	O
		15	Noir	O	O
37	FLEUR_JAUNE	-	-	O	O
38	FLEUR_CYAN	-	-	O	O
41	BLOC_D_OR	-	-	O	O
42	BLOC_DE_FER	-	-	O	O
45	BRIQUE	-	-	O	O
46	TNT	0	Inactif	O	O
		1	Prêt à exploser	O	O

Tableau 10 *suite*

ID	Constante	Valeur de donnée	Sous-catégorie	Pi	PC/ MAC
49	OBSIDIENNE	-	-	O	O
50	TORCHE	0	Posée sur le sol	O	O
		1	Tournée vers l'est	O	O
		2	Tournée vers l'ouest	O	O
		3	Tournée vers le sud	O	O
		4	Tournée vers le nord	O	O
53	ESCALIER_EN_BOIS	0	Tourné vers l'est	O	O
		1	Tourné vers l'ouest	O	O
		2	Tourné vers le sud	O	O
		3	Tourné vers le nord	O	O
		4	Tourné vers l'est (à l'envers)	O	O
		5	Tourné vers l'ouest (à l'envers)	O	O
		6	Tourné vers le sud (à l'envers)	O	O
		7	Tourné vers le nord (à l'envers)	O	O
57	BLOC_DE_DIAMANT	-	-	O	O
80	BLOC_DE_NEIGE	-	-	O	O
89	PIERRE_LUMINEUSE	-	-	O	O
246	OBSIDIENNE_LUMINEUSE	-	-	O	N
247	REACTEUR_DU_NETHER	0	Inutilisé	O	N
		1	Actif	O	N
		2	Inactif/épuisé	O	N

Tableau 11 – L'API MinecraftStuff (MinecraftDrawing)	
Importer l'API MinecraftStuff	**Créer l'objet MinecraftDrawing**

`import mcpi.minecraftstuff as ↵`
`minecraftstuff`
	`mc = minecraft.Minecraft.create()`
	`mcdrawing =`
	`minecraftstuff.MinecraftDrawing(mc)`

Tracer un segment entre deux points	**Obtenir toutes les positions des blocs d'une ligne sous la forme d'une liste**

`mcdrawing.drawLine(0, 0, 0, 10, 10, 10, block.DIRT.id)`	`ligne = mcdrawing.getLine(0,0,0,10, 10,10)`
	`pos1 = ligne[0]`
	`print(pos1.x)`

Dessiner une sphère	**Tracer un cercle**
`mcdrawing.drawSphere(0, 0, 0, radius, block.DIRT.id)`	`mcdrawing.drawCircle(0, 0, 0, radius, block.DIRT.id)`
Dessiner un polygone plat (par exemple : un triangle)	

`tri = []`
`filled = True`
`tri.append(minecraft.Vec3(0,0,0))`
`tri.append(minecraft.Vec3(10,0,0))`
`tri.append(minecraft.Vec3(5,10,0))`
`mcdrawing.drawFace(tri, filled,`
`block.DIRT.id)`	

Tableau 12 – L'API MinecraftStuff (MinecraftShape)	
Créer une forme MinecraftShape	**Dessiner et effacer une forme**

`mc = minecraftmMinecraft.create()`
`blocs =`
`[minecraftstuff.ShapeBlock(0,0,0, block.DIRT.id),`
`minecraftstuff.ShapeBlock(1,0,0, block.DIRT.id)]`
`pos = Vec3(0,0,0)`
`forme =`
`minecraftstuff.MinecraftShape(mc,`
`pos, blocs)`
	`shape.draw()`
	`shape.clear()`

Repositionner une figure	**Déplacer une figure d'un certain nombre de blocs**
`shape.move(10, 10, 10)`	`ymove = 1` `shape.moveBy(0, ymove, 0)`

Glossaire

API (chapitre 2) – Correspond à *Application Programming Interface* (« interface de programmation », en français). Une API te permet d'accéder en toute sécurité à certaines parties du programme d'une application depuis tes propres programmes. En l'occurrence, c'est par l'API de Minecraft que tu peux accéder au jeu dans tes programmes Python.

Appel (chapitre 4) – Lorsque tu effectues un appel de fonction, Python se souvient de l'endroit où il s'arrête dans ton script et plonge temporairement au sein de la fonction, à partir de `def`, qui marque le début de ta fonction. Lorsqu'il arrive à la fin de la fonction, Python fait un saut en arrière, là où il s'y est introduit.

Booléen (chapitre 6) – Une variable qui ne peut détenir que deux valeurs possibles, soit `True` (vrai) soit `False` (faux).

Boucle imbriquée (chapitre 6) – Une boucle à l'intérieur d'une autre boucle. On appellera la première une `boucle externe`, et la deuxième, une **boucle interne**. En Python, on imbrique une boucle dans une autre en l'indentant d'un niveau supplémentaire. On peut imbriquer plusieurs boucles les unes dans les autres autant de fois qu'on le souhaite.

Boucle infinie (chapitre 2) – Un type de boucle qui ne s'arrête jamais. Elle tourne à l'infini. L'unique façon de l'arrêter est de mettre fin au programme Python lui-même.

Bukkit (chapitre 1) – Un serveur Minecraft qui permet aux utilisateurs de modifier le jeu à l'aide de modules.

Chaîne (chapitre 2) – Une variable qui permet de stocker toute une suite de lettres, de nombres et de symboles. En clair, c'est comme si vous enfiliez des perles sur un collier ou un bracelet, chaque perle étant ornée d'un nombre, d'une lettre ou d'un symbole différent, le tout dans ordre bien défini et toujours identique.

Constante (chapitre 2) – Comme pour une variable, une constante est le nom qu'on donne à un fragment de mémoire, dans lequel on peut stocker des valeurs qui restent identiques à chaque fois que le programme s'exécute.

Coordonnée (chapitre 2) – Un ensemble de nombres qui indiquent une position. Dans Minecraft, les coordonnées 3D sont utilisées pour indiquer la position exacte

de ton joueur dans le jeu en trois dimensions. Chaque coordonnée contient trois nombres.

Coordonnées absolues (chapitre 3) – Une série de coordonnées qui utilisent des valeurs fixes pour définir la position d'un point donné, comme celle d'un bloc Minecraft par exemple.

Coordonnées relatives (chapitre 3) – Une série de coordonnées définies par rapport à un autre point, par exemple par rapport à la position de ton joueur dans Minecraft.

Courant (chapitre 5) – Le taux d'énergie électrique qui circule au-delà d'un point dans un circuit. C'est l'équivalent du débit d'eau dans un tuyau. Il se mesure en **ampères** (A). Pour les courants plus faibles, on utilisera le **milliampère** (mA) comme unité de mesure.

CSV (chapitre 6) – (qui correspond à *Comma Separated Values*) Un type de fichier de texte dans lequel on enregistre des valeurs séparées par des virgules.

Face (chapitre 7) – L'une des surfaces planes qui composent un objet, par exemple le côté d'un cube ou le haut d'un tambour.

Fin de ligne (chapitre 6) – Un caractère spécial invisible que l'ordinateur utilise pour marquer la fin d'une ligne de texte.

Fonction (chapitre 3) – Au sein d'une fonction Python, tu peux rassembler plusieurs instructions et leur donner un nom spécifique. Lorsque tu auras besoin d'exécuter ces instructions, tu n'auras plus qu'à écrire le nom de la fonction, suivi de parenthèses.

Géorepérage (chapitre 2) – Une technique qui consiste à encercler des coordonnées sur une carte.

Global (chapitre 3) – Une variable globale est une variable qu'on peut utiliser partout dans le programme. Toute variable (ou constante) n'étant pas indentée à gauche est une variable globale, à laquelle on peut accéder n'importe où.

GPIO (chapitre 5) – Les broches GPIO permettent de recevoir ou d'émettre des signaux vers ou depuis le processeur central d'un système informatique. Elles ont été conçues pour répondre à un usage général et les concepteurs de circuits peuvent leur assigner une fonction précise, telle que le contrôle de circuits externes.

IDLE (chapitre 1) – Un éditeur de code et débogueur Python.

Indentation (chapitre 2) – L'indentation est l'espace qui se situe à gauche, au début de tes lignes de programmation. Les espaces d'indentation permettent de structurer le programme et de regrouper plusieurs instructions en blocs distincts, sous des

boucles ou sous d'autres instructions. En Python, les niveaux d'indentation sont importants, car ils permettent aussi de modifier le sens des programmes.

Index (chapitre 4) – Un nombre qui permet d'accéder à un objet en particulier dans une liste donnée. Pour signaler un index, tu dois utiliser des crochets, comme cela : `a[0]` représente le premier objet de la liste et `a[1]` le second. Les index en Python commencent toujours par le chiffre 0. 0 désigne donc toujours le premier objet d'une liste.

Instruction (chapitre 2) – Un terme informatique général qui signifie que tu commandes à ton ordinateur d'effectuer une action complète, en saisissant par exemple une ligne de programmation comme `print("Bonjour Steve")`.

Interface (chapitre 2) – Un ensemble de règles qui indiquent comment, en tant que programmeur, tu peux accéder à certaines parties du système d'exploitation d'un ordinateur.

LED (chapitre 5) – Une diode électroluminescente est un type de diode qui s'allume lorsqu'un courant électrique la traverse. Une LED n'autorise le courant à passer que dans un seul sens et ne s'allume que s'il passe dans le bon sens.

Liste (chapitre 4) – Un type de variable qui permet d'entreposer plusieurs données dans un seul et même endroit. Parmi toutes les possibilités qu'elle offre, tu peux notamment y ajouter de nouveaux objets, mesurer sa longueur, accéder aux objets à des endroits précis et les supprimer, quel que soit leur emplacement dans la liste.

Métacaractère (chapitre 6) – Un caractère spécial qui permet de sélectionner plusieurs noms ou mots identiques. Il fonctionne un peu comme un « joker » dans un jeu de cartes, car il peut représenter tout et n'importe quoi.

Métadonnées (chapitre 6) – Des données sur des données. Si ton fichier de données représente un objet Minecraft, les métadonnées décriront par exemple les dimensions de l'objet, son nom ou le nom de son créateur.

Nombre aléatoire (chapitre 3) – Un nombre qui provient généralement d'une séquence de nombres aléatoires, à savoir d'une liste de nombres n'étant fondée sur aucun modèle prédéfini ni sur aucune séquence répétitive.

Paramètre (chapitre 7) – Pour s'exécuter, une fonction a besoin d'informations, comme `setBlock()` qui requiert des coordonnées x, y, z et un type de bloc. Ces informations sont ce qu'on appelle des **paramètres**. Lorsqu'un programme utilise cette fonction, ont dit qu'il **l'appelle** et qu'il lui **passe** des valeurs en paramètres.

Plaque d'essais (chapitre 5) – Un outil réutilisable qui te permet de monter des circuits sans avoir recours à la soudure. Les plaques d'essais contiennent plusieurs rangées de trous dans lesquels tu peux brancher des fils, des câbles et des composants afin de créer des circuits.

Plug-in (chapitre 1) – Un programme qui fonctionne au sein du serveur Bukkit et permet de modifier Minecraft.

Pop (chapitre 4) – Lorsque tu appliques la méthode pop à une liste, elle en supprime le dernier élément.

Prévisible (chapitre 8) – Lorsqu'il est possible d'anticiper un événement qui va se produire, il est prévisible.

Probabilité (chapitre 8) – Un calcul qui permet de connaître la fréquence d'un événement. Par exemple, lorsqu'on joue à pile ou face, il y a 50 % de chances (1 sur 2) de tomber sur le côté face.

Programme (chapitre 2) – Une série d'instructions que tu saisis dans un langage informatique spécifique et que l'ordinateur respectera à la lettre.

Prototype (chapitre 5) – Un modèle réduit d'un objet conçu pour réaliser des tests de fonctionnement.

Python (chapitre 1) – Le langage de programmation qui est utilisé dans cet ouvrage.

Racine carrée (chapitre 8) – La racine carrée d'un nombre est la valeur qui, multipliée par elle-même, permet d'obtenir le nombre en question. Par exemple, la racine carrée de 9 est 3, car $3 \times 3 = 9$.

Résistance (chapitre 5) – Un composant électrique qui permet de résister au courant dans un circuit.

Return (chapitre 6) – Une instruction qui permet de renvoyer la valeur d'une fonction lorsqu'elle retourne dans le programme qui utilise la fonction pour la première fois.

Sensible à la casse (chapitre 1) – Python est un langage de programmation sensible à la casse, c'est-à-dire qu'il faut saisir le code en respectant les minuscules et les majuscules.

Séquentiel (chapitre 9) – Qui suit un ordre ou une séquence, c'est-à-dire qui exécute des instructions les unes à la suite des autres.

Syntaxe (chapitre 2) – Fait référence aux règles d'un langage (en l'occurrence, le langage Python) et à l'ordre dans lequel tu saisis les informations.

Tension (ou **voltage**) (chapitre 5) – La différence d'énergie électrique qui existe entre deux points d'un même circuit. Elle est semblable à la pression qu'exerce l'eau dans les tuyaux de canalisations ; c'est cette même pression qui pousse le courant à s'écouler dans un circuit. Elle se mesure en **volts** (V).

Trigonométrie (chapitre 7) – Une branche des mathématiques qui traite des relations entre distances et angles dans les triangles.

Variable (chapitre 2) – Une donnée que tu stockes dans la mémoire de ton ordinateur. Tu peux attribuer de nouvelles valeurs à tes variables n'importe où dans tes programmes. Tu peux également les rappeler et les afficher ou les utiliser dans des calculs.

Variable de comptage (chapitre 3) – La boucle `for` est appelée une **variable de comptage** ou une **variable de comptage contrôlé,** car elle s'exécute un nombre limité de fois.

Index

Symboles

% (modulo) 314

A

API
  définition 331
  Minecraft 326
    blocs 327, 328, 329
  MinecraftStuff
    MinecraftDrawing 330
    MinecraftShape 330
appeler une fonction
  définition 331

B

bloc
  API Minecraft 327, 328, 329
booléen
  définition 331
boucle
  imbriquée
    définition 331
  infinie
    définition 331
  Python 322
brancher un dispositif électronique 325
Bukkit
  définition 331

C

chaîne
  définition 331
conditions
  Python 322
constante
  définition 331
coordonnée
  absolue
    définition 332
  définition 331
  relative
    définition 332
courant
  définition 332
CSV
  définition 332

E

échafaudage 311
EDI. Voir IDLE

F

face
  définition 332
fin de ligne
  définition 332
fonctions
  définition 332
  Python 323

G

géorepérage
  définition 332
GPIO
  définition 332

I

IDLE
  définition 332
indentation
  définition 332
index
  définition 333
instruction
  définition 333
  return 334
interface
  définition 333

L

LED
  définition 333
liste
  définition 333
  Python 323

M

métacaractère
  définition 333
métadonnées
  définition 333
Minecraft
  API 326
MinecraftStuff
  API 330
module
  Python 324
modulo (opérateur) 314

N

nombre
  aléatoire
    définition 333

P

paramètre
  définition 333
plaque d'essais
  définition 333
plug-in
  définition 334
pop
  définition 334
prévisible (définition) 334
probabilité (définition) 334
programmation
  parallèle. Voir multithreading
programme (définition) 334

prototype
définition 334
Python
boucles 322
conditions 322
fonctions 323
listes 323
modules 324

R

racine carrée (définition) 334
résistance
définition 334
retour chariot. Voir fin de ligne
return (instruction)
définition 334

S

scaffolding 311
sensible à la casse (définition) 334
séquentiel (définition) 334
syntaxe
définition 334

T

tension ou voltage
définition 334
traitement de fichiers 324
trigonométrie
définition 335

V

variable
de comptage
définition 335
définition 335
globale
définition 332
variables
d'entrée et de sortie 321