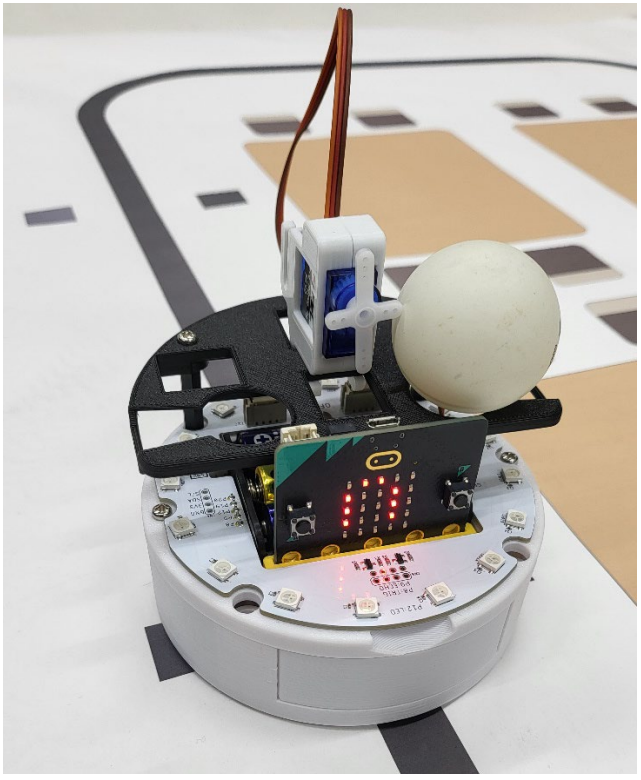
配膳ロボプリント 2022

（基本編）



注意事項

このプリントはなくさずに，毎回持ってくること！

＿＿＿＿組＿＿＿＿番　氏名＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

はじめに

このプログラムは、70 行ほどから成り立ちます。大きいプログラムになると、上から順番に書いて、実行していくことはできないので、作成順序を分割し、動作行為を小目標としてクリアしていく方法を取ろうと思います。

予定

まずは，配膳ロボットを 5 コマ程度使い，配膳ロボの基本動作のプログラムを書いていきます。それが終わったら，5 コマ程使い，その配膳ロボットをカスタマイズさせて，自分達のオリジナル配膳ロボを作り，会社の社長，副社長になりきってプレゼンテーションを行います。

基本編　5 コマ程度，カスタマイズ編　5 コマ程度

基本編作成順序

作成順序は以下の通りです。

STEP1.　コードはコメントアウトを使い行為を分解しておく。
STEP2.　モジュールを読み込み、まっすぐ走る。
STEP3.　黒い線まで走っていき，黒い線になったらとまる。
STEP4.　STEP3 のプログラムをコメントアウトし，ライントレースする関数を書いてライントレースする。
STEP5.　マーカーの長さが 180 以上になったら（キッチンにいったら），とまる。
STEP6.　マーカーの長さが 50 以上なら，テーブルをカウントしていく。
STEP7.　テーブル番号のセットをする。
STEP8.　もし，配膳テーブルと一緒なら，とまり，配膳する。(完成)
STEP9.　(発展編) 動き始めたら，青く光り，配膳後は赤く光り，キッチンについたら消える。
STEP10.　(発展編) 動き始めたら，青く点滅し，配膳後は赤く点滅し，キッチンについたら消える。
STEP11.　(発展編) キッチンについたら，時計回りに青い光が移動し，青い光を一周させる。
STEP12.　(発展編) 動き始めたら，時計回りに青い光が移動し，配膳後は反時計回りに赤い光が移動し，キッチンについたら消える。

注意事項

- **コメント以外はすべて半角で入力！**
- **スペースは全角スペースを使ってはいけない！**
- **字下げする場合、tab 一つ分か、半角スペース 4 つ分を開ける！**

  ※コロン(:)をうって、エンターキーを押せば、自動的に字下げがされます。

STEP１　コードはコメントアウトを使い行為を分解しておく。

プログラムの全体像は以下のようになっています。だいたいは上から順番に書いていきますが，上に戻ったり，下に飛んだりし，コードを記入するので，行番号がコロコロ変わります。そのため，コードの部分をある程度は分類わけし，その中でコーディングをする方法をとります。まずは，コメントアウト(#)を先頭に入力すれば，そのあとはプログラムとしては無効化されるので，まずは以下の分類わけを記述してください。

#①モジュールの読み込み，初期化，インスタンス化など#

#②ライントレースの関数

#③テーブル番号のセットプログラム

#④メインプログラム

合格判定

上記を撮った写真を見せに来る。

よりプログラミングが理解できるコラム　①オブジェクト指向プログラミングの考え方

処理の流れに沿ってプログラミングしていくことを手続き型プログラミングという。しかしこれでは、同じ行為を繰り返す際に、もう一度コードを書かなければいけなかったり、コードをひとまとまりにできなかったりと効率が悪い場合がある。そこで、関連するデータ（変数）とそれに対する操作（関数）をまとめて、一つのモノ（オブジェクト）として捉え、プログラミングしていくことをオブジェクト指向プログラミングという。

haizen.py を読み込み，以下のコードを記述する。

コード部分	解説
#①モジュールの読み込み，初期化，インスタンス化など# from microbit import * from haizen import * motor = WheelMotor()	 配膳ロボを動かすためのコード集を読み込む 音楽再生のためのモジュールを読み込む モジュールからインスタンス（オブジェクト）を作る。

解説：

まず初めに，今回のプログラミングに必要なファイルを呼び出し，モーターを動かす「モノ」を作ります。

コード部分	解説
#④メインプログラム while True: 　motor.set_speed(30,30)	 ずっと繰り返す 　モーターのスピードを左右 30,30 にする。

ここで走らせに行く

合格判定

まっすぐ走ったら合格。少しずれる場合，30,30 を微調整してもよい。

※調整方法

右に曲がっていく場合，左側を小さくし，右側を大きくする。例：motor.set_speed(28,32)

左に曲がっていく場合，左側を大きくし，右側を小さくする。例：motor.set_speed(32.28)

よりプログラミングが理解できるコラム　②インスタンスとは何か　motor = WheelMotor()　って何？

Python のようなオブジェクト指向プログラミングでは，設計（クラス）とそれによって作られたモノ(インスタンス)を区別してコーディングしていく。モジュールとして読み込んだファイルがクラス（設計図）のかたまりで，motor = WheelMotor()という部分をインスタンス化，motor という部分をインスタンス（またはオブジェクト）という。

なぜそのようにするかというと，まとまりのあるプログラミングを実現できるからである。例えば，モーターを動かすというプログラムには，100 行ほどのプログラムから成り立つ。その中で，まっすぐ進むというコードを書いた際，少なくとも数十行はコードを要す。これをメインプログラムに記述していった際に，大変見にくくなってしまう。これらは別ファイル（モジュール）にして、作っておいて、それらを呼び出していけば見やすいコードになる。一度，100 行のコードをモノとしてインスタンス化しまえば，あとはそのモノをどう扱っているかをコーディングしていけばいいので、コードの見通しがよく、わかりやすくなる。

haizen.py（全部で約 300 行ぐらいある）というファイルの中に・・・

```
class WheelMotor:                    #ホイールモータークラス
    def __init__(self,):
        self.enc_count_l = 0
        self.enc_count_r = 0
・・・(略)・・・
    def set_speed_l(self, speed_l):
        if speed_l < -100:
            speed_l = -100
        elif speed_l > 100:
            speed_l = 100
        if speed_l >= 0:
            pin13.write_digital(0)
            pin14.write_analog((1023 / 100) * speed_l)
        else:
            pin13.write_digital(1)
            pin14.write_analog((1023 / 100) * (-speed_l))
            pin16.write_analog((1023 / 100) * (-speed_r))
・・・(略)・・・
    def set_speed(self, speed_l, speed_r):
        self.set_speed_l(speed_l)
        self.set_speed_r(speed_r)
・・・(略)・・・
class LineSensor:                    #ラインセンサークラス
    （続く）
```

中身をみると，WheelMotor クラスというものがあり，その中に様々な動きが事細かに書かれている。これを motor というインスタンス（またはオブジェクト）にして，利用している。

～インスタンスとオブジェクトの違い～

包含関係でいうと，オブジェクトの方が広い。オブジェクトの中にインスタンスがある。インスタンス化されていないオブジェクトもある。この場合，オブジェクトという。

STEP３　黒い線まで走っていき，黒い線になったらとまる。

コード部分	解説
#④メインプログラム **while True:** 　　**motor.set_speed(30,30)** 　　sensor1 = pin1.read_analog() 　　sensor2 = pin2.read_analog() 　　if sensor1 >=200 and sensor2 >= 200: 　　　　motor.set_speed(0,0) 　　　　break	 黒いラインを挟む二つのセンサーの値を読み取る。 もし，両方とも 200 以上なら， モータをとめ， 繰り返しを抜ける。

配膳ロボットの電源は，コースにいってから入れること。

合格判定

黒い線までいったらとまる。距離をかえて二回試すこと。

STEP4　STEP3 のプログラムをコメントアウトし，ライントレースする関数をかいて，ライントレースする。

コード部分	解説

#②ライントレースの関数
def trace_line():
　　sensor1 = pin1.read_analog()
　　sensor2 = pin2.read_analog()
 	
　　if sensor1 < 200 and sensor2 < 200:
　　　　motor.set_speed(30,30)
　　elif sensor1 < 200 and sensor2 >= 200:
　　　　motor.set_speed(30, 0)
　　elif sensor1 >= 200 and sensor2 < 200:
　　　　motor.set_speed(0, 30)
　　else:
　　　　motor.set_speed(0,0)
	
	ライントレース関数の定義
	左側のセンサーのアナログ値を取得
	右側のセンサーのアナログ値を取得
	
	両方とも白であれば
	まっすぐ進む
	左側が白で，右側が黒なら
	左に移動する。
	左側が黒で，右側が白なら
	右に移動する。
	それ以外（両方とも黒）なら
	とまる。

コード部分	解説
#④メインプログラム **while True:** 　　# motor.set_speed(30,30) 　　# sensor1 = pin1.read_analog() 　　# sensor2 = pin2.read_analog() 　　# if sensor1 >=200 and sensor2 >= 200: 　　#　　motor.set_speed(0,0) 　　#　　break 　　trace_line()	 STEP3 のものはコメントアウトしておく。 ここまでコメントアウト ライントレース関数を記述する。

配膳ロボットの電源は，コースにいってから入れること。

合格判定

コースを一周する。

STEP5　マーカーの長さが 180 以上になったら（キッチンにいったら），とまる。

コード部分	解説
while True: 　　**# motor.set_speed(30,30)** 　　**# sensor1 = pin1.read_analog()** 　　**# sensor2 = pin2.read_analog()** 　　**# if sensor1 >=200 and sensor2 >= 200:** 　　**#　　motor.set_speed(0,0)** 　　**#　　break** 　　**trace_line()**	
marker_len = find_marker()	マーカーの長さを測定する関数を書き,
if marker_len > 180:	その長さが 180 以上なら（ここは調整の可能性あり）
motor.set_speed(0,0)	とめて
display.scroll('Kitchen')	キッチンと表示し
reset()	リセット関数を書いて，電源時も戻る

配膳ロボットの電源は，コースにいってから入れること。

※180 の長さは電池の残量によって長くしたり，短くしたりする場合あり。

合格判定

キッチンにきたら停止する。

STEP6　テーブルをカウントするプログラムを書く。

コード部分	解説
#①モジュールの読み込み，初期化，インスタンス化など# from microbit import * from haizen import * motor = WheelMotor() table_count = 0	テーブルカウントを初期化する。

コード部分	解説

#④メインプログラム
while True:
　# motor.set_speed(30,30)
　# sensor1 = pin1.read_analog()
　# sensor2 = pin2.read_analog()
　# if sensor1 >=200 and sensor2 >= 200:
　#　　motor.set_speed(0,0)
　#　　break
　trace_line()
　marker_len = find_marker()
　if marker_len > 180:
　　motor.set_speed(0,0)
　　display.scroll('Kitchen')
　　reset()
　elif marker_len > 50:
　　table_count = table_count + 1
　　display.show(table_count, wait = False)
	その長さが 50 以上なら（ここは調整の可能性あり）
	テーブルのカウントを 1 増やして
	ディスプレイにずっと表示。

合格判定

環状線上から始まり，テーブルマーカーごとにカウントがあがっていけば合格

STEP7　テーブル番号のセットをする。

コード部分	解説
#①モジュールの読み込み，初期化，インスタンス化など# **from microbit import *** **from haizen import *** **motor = WheelMotor()** **table_count = 0** target_table = 0 max_table = 4	 配膳先のテーブルを初期化 テーブル数を設定

コード部分	解説

#③テーブル番号のセットプログラム
display.show(target_table)
 	
while True:
　if button_a.was_pressed():
　　target_table += 1
　　if target_table > max_table:
　　　target_table = 1
　　display.show(target_table)
　if button_b.was_pressed():
　　if target_table == 0:
　　　display.show(Image.NO)
　　else:
　　　display.show(Image.YES)
　　　sleep(1000)
　　　break
	
	配膳先のテーブルを表示。初期値は 0
	
	a ボタンがおされたら，
	配膳先のテーブルを 1 プラスし，
	もし，配膳先のテーブルがテーブル数以上なら，
	1 に戻す。
	配膳先のテーブルを表示
	b ボタンがおされたら，
	もし，配膳先のテーブルが 0 のままだったら（a が押されていなければ）
	×と表示させ，
	そうでなければ，
	✓
	と 1 秒表示させ，
	繰り返しから抜ける。

合格判定

b ボタンを押したら，×になり，a ボタンを押したら，カウントが増えていき，4 の次は 1 に戻り， b ボタンを押したら，チェックマークが 1 秒現れれば OK。

コード部分	解説
#①モジュールの読み込み，初期化，インスタンス化など#	
from microbit import *	
from haizen import *	
motor = WheelMotor()	
table_count = 0	
target_table = 0	
max_table = 4	
serving = True	配膳中フラグを真にする。
servo = Servo180(pin8)	サーボ 180 度モーターは pin8 に刺している
servo.set_angle(90)	最初の角度を 90 度にする。

コード部分	解説
#④メインプログラム	
while True:	
# motor.set_speed(30,30)	
# sensor1 = pin1.read_analog()	
# sensor2 = pin2.read_analog()	
# if sensor1 >=200 and sensor2 >= 200:	
# motor.set_speed(0,0)	
# break	
trace_line()	
marker_len = find_marker()	
if marker_len > 180:	
motor.set_speed(0,0)	
display.scroll('Kitchen')	
reset()	
elif marker_len > 50:	
table_count = table_count + 1	
display.show(table_count, wait = False)	
if table_count == target_table:	もし，テーブルのカウントと，配膳先のテーブルが同じなら
motor.set_speed(0,0)	2 秒停止し，
sleep(2000)	
servo.set_angle(0)	サーボモータを 0 度にして，
sleep(2000)	2 秒停止し，
servo.set_angle(90)	再びサーボモータを 90 度にして
serving = False	配膳フラグを偽にする。

合格判定

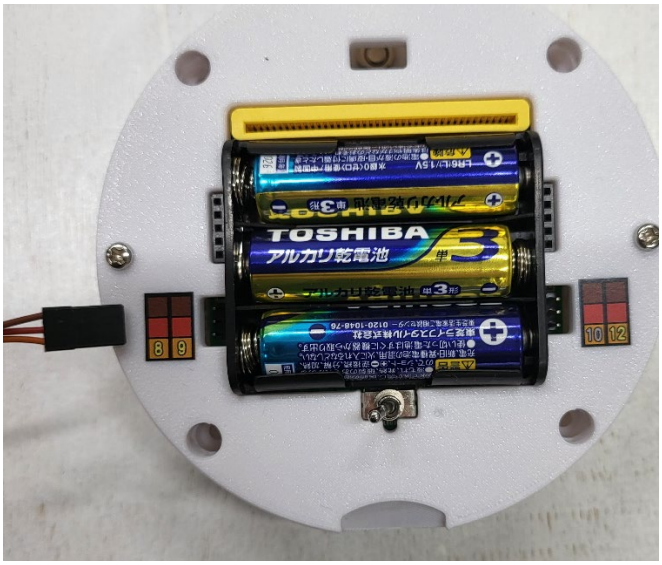
指定されたテーブルに配膳できたら OK。

サーボモータは以下のように使ってください。

モータ

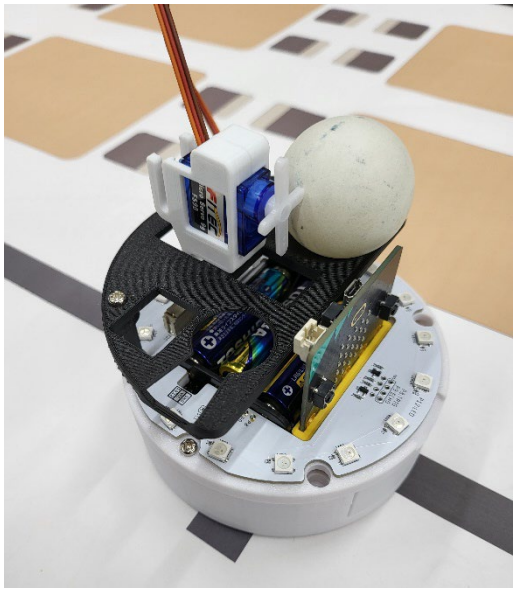
※十字のパーツを差し込みます

差す位置



※左の外側(pin8)に刺してください。茶色い線が進行方向側です。

以下のようにサーボモータをつけ，ピンポン玉をおいてください。



クリアできたら

クリアおめでとうございます！

下の save ボタンをおし， complete というファイル名で保存し，ダウンロードフォルダにダウンロードされるので，自分のフォルダ（R:）に移動しておきましょう。

STEP9　（発展編）動き始めたら，青く光り，配膳後は赤く光り，キッチンについたら消える。

まず，円形の LED を装着します。
次に以下のコードを追加します。

コード部分	解説
#①モジュールの読み込み，初期化，インスタンス化など **from microbit import *** **from haizen import *** import neopixel np = neopixel.NeoPixel(pin12,16) #（以下略）	 neopixcel を使うことを宣言 neopixcel を pin12 にさして，ピクセル数が 16 であることを宣言

#④メインプログラム　の中で

np.fill((0, 0, 100))
np.show()

とすれば，LED を全部青に光らせることができます。かっこの中は第一引数（一番左）は赤，第二引数（真ん中）は緑，第三引数（一番右）は青でそれぞれの光の強さを表しています。

消す場合は,
np.fill((0,0,0))
np.show()
あるいは
np.clear()

とすれば消すこともできます。これ適切な場所に追加していきましょう。

ヒント 1 ：serving 変数に関して，初期化で True にしておいて，配膳が終わったら False にしています。この変数を使って挙動を考えます。
ヒント 2：if serving == True:　または if serving:
とどこかに書き，配膳中は青，配膳終わったら，赤にしましょう。

STEP10（発展編）動き始めたら，青く点滅し，配膳後は赤く点滅し，キッチンについたら消える。

配膳ロボットでは，動いている最中は sleep 関数は使うことができません。（もちろん，とまっている最中は sleep 関数が使えます。）そこで，動いている最中に，挙動を変えていきたい場合，割り込みプログラムを書いていきます。

コード部分	解説
#①モジュールの読み込み，初期化，インスタンス化などの中に以下を追加。	
record_time = 0	記録時間
TIME_INTERVAL = 500	何ミリ秒間隔で変えていきたいか
motion_number = 0	動作番号を 0 に初期設定

コード部分	解説
#④メインプログラム	trace_line()関数のすぐ下に割り込みプログラムを書いていく。
while True:	
# motor.set_speed(30,30)	
# (略)	
# break	
trace_line()	
if running_time() - record_time > TIME_INTERVAL:	動いている時間から記録時間をひき，それが TIME インターバル以上であれば，記録時間を動いている時間に上書き。
record_time = running_time()	
if motion_number == 0:	もし，動作番号が 0 番なら
#ここにしたい処理	したい処理を記述し
motion_number = 1	動作番号を 1 番にしておく。
elif motion_number == 1:	もし，動作番号が 1 なら
#ここにしたい処理	したい処理を記述し，
motion_number = 0	動作番号を 0 番にしておく。

このようなプログラムを書くことで，２つの動作を交互に実行させるようなプログラムが作れます。点灯と消灯のプログラムを「#ここにしたい処理」の中に記述すれば，課題を解くことができます。このようにすることで 500 ミリ秒間隔で動作番号 0 の行為と動作番号 1 の行為を交互に実行させることができます。

STEP１１（発展編）キッチンについたら，時計回りに青い光が移動し，青い光を一周させる。

一つの LED だけ光らせる場合は配列（リスト）を用います。例えば，0 番目を光らせたいのであれば,

```
np[0] = (0,0,100)
np.show()
```

とすれば，0 番の LED のみ光らせることができます。
他にも

```
i=1
np[i]=(0,0,100)
np.show()
```

とすれば，1 番の LED を光らせることができます。
for 文や while 文を使ってぐるっと光がまわるプログラムを作りましょう。

STEP１２（発展編）動き始めたら，時計回りに青い光が移動しつづけ，配膳後は反時計回りに赤い光が移動しつづけ，キッチンについたら消える。

最終問題です。これはノーヒントでお願いします。