プログラミング教室のテクノロ

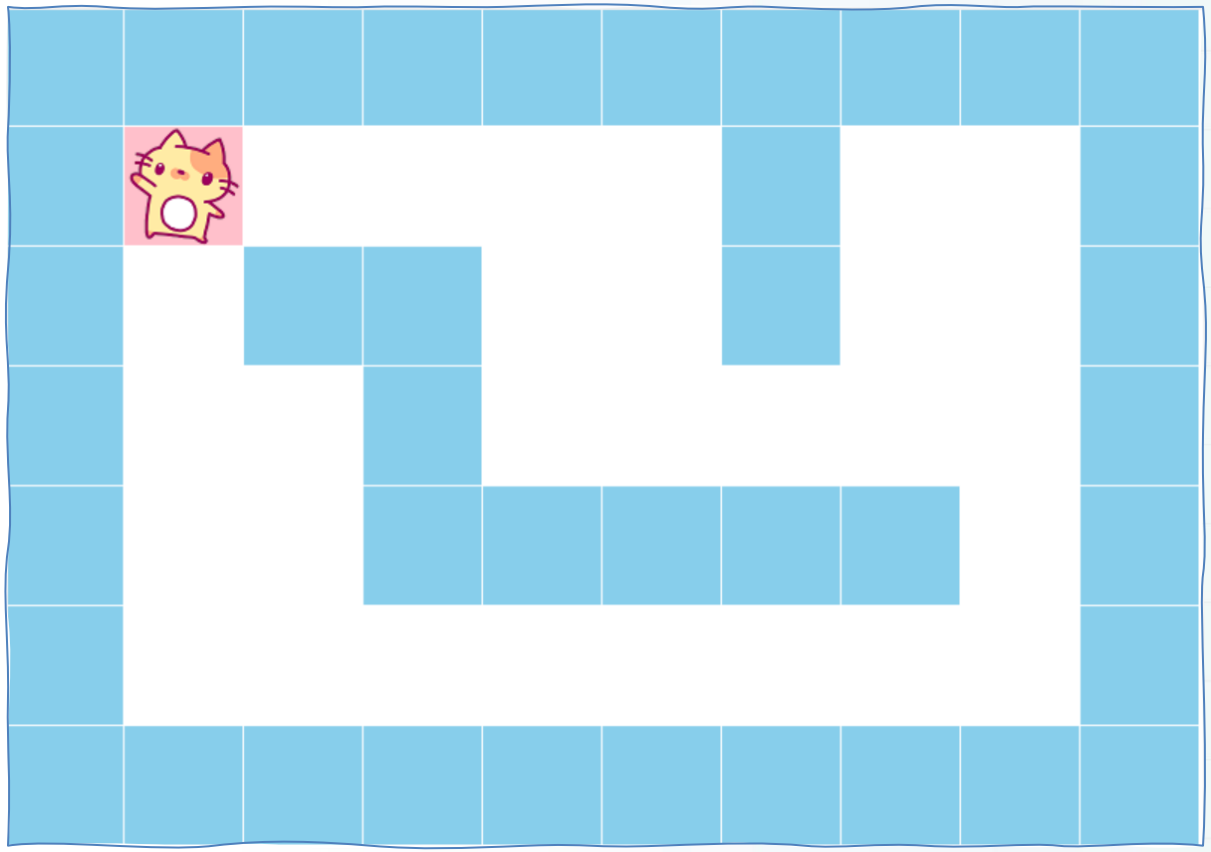
Pythonの道

本格的なゲーム開発①(後半)

もくじ

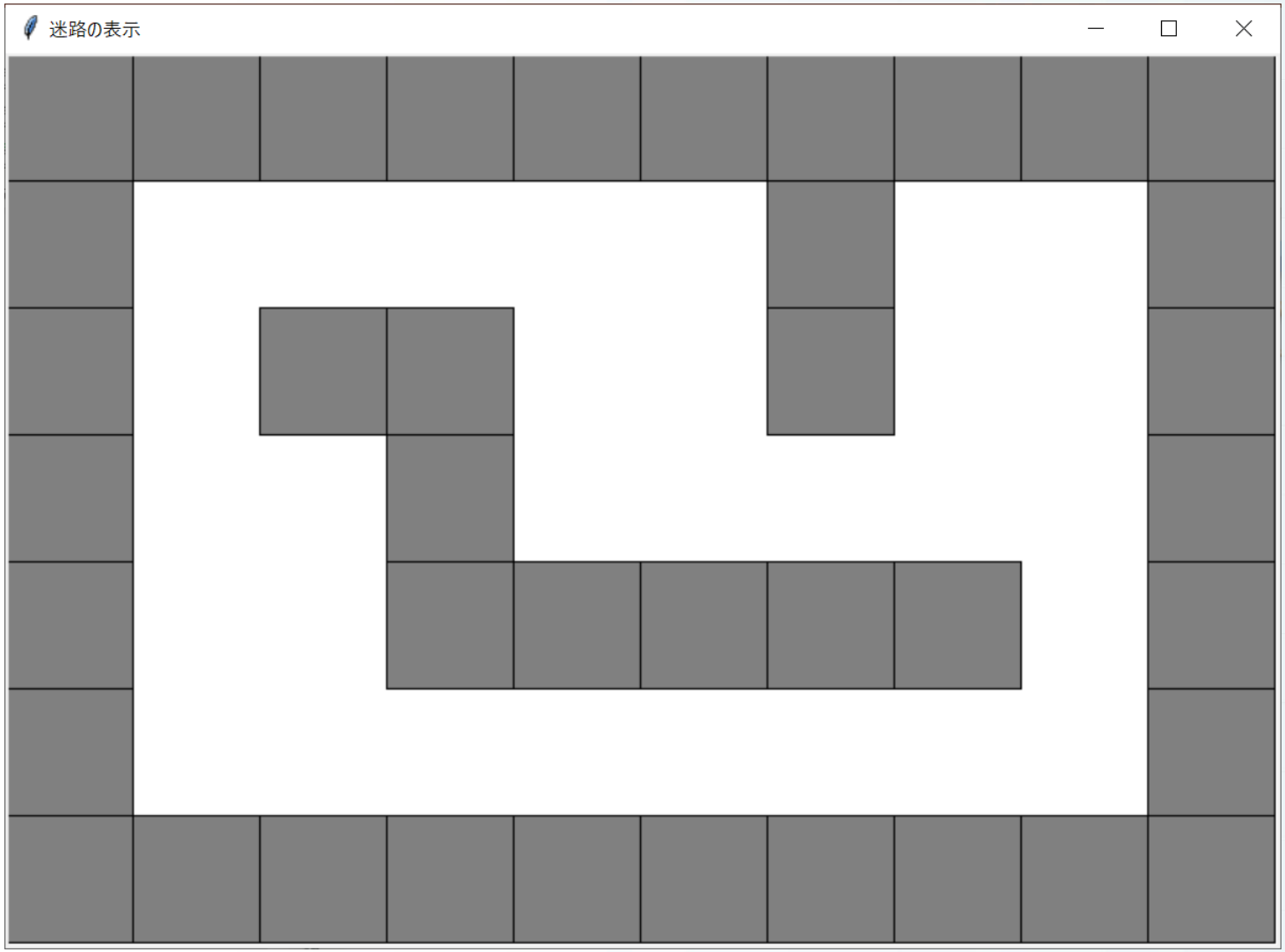
・床塗りゲームをつくる（後半）

二次元リストを使って迷路データを定義する方法を学ぶ



迷路のデータを定義する

二次元の画面構成のゲームでは背景のデータをリストで管理する。リストで迷路を定義し、ウィンドウに表示してみよう。



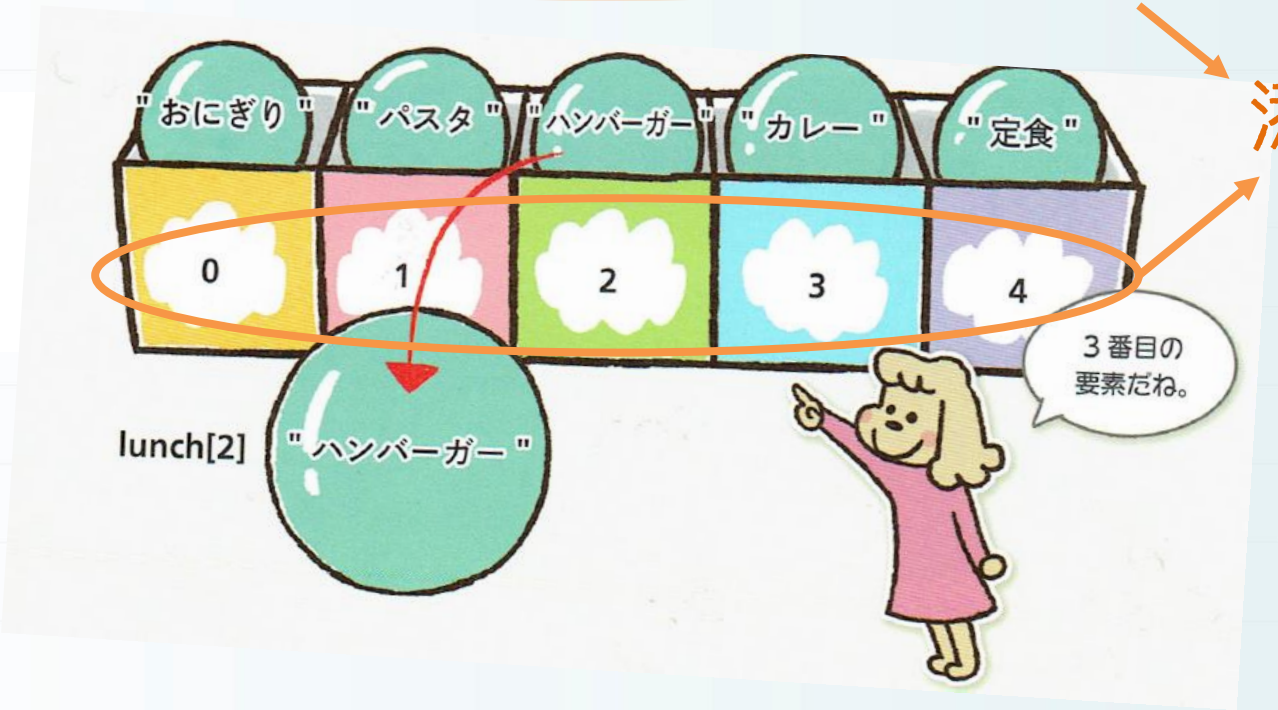
二次元リストについて

迷路のようなデータは二次元のリストで定義する。二次元リストとは横方向(行）と縦方向（列）に添え字を使ってデータを扱うリストのことをいう。

添え字についてのおさらい

```
lunch = [" おにぎり" ," パスタ" ," ハンバーガー" ," カレー" ," 定食" ]
          0          1          2          3          4
print(lunch[2])
```

添え字



ハンバーガー

>>>

二次元リストについて

横方向をx、縦方向をyとした場合の添え字の表し方を理解する。

m[y][x]

添え字

x →

y ↓

m[0][0]	m[0][1]	m[0][2]	m[0][3]
m[1][0]	m[1][1]	m[1][2]	m[1][3]
m[2][0]	m[2][1]	m[2][2]	m[2][3]

例えば右下角のm[2][3]に10を代入する場合、

m[2][3] ＝ 10 と記載する。

1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	0	0	1
1	0	1	1	0	0	1	0	0	1
1	0	0	1	0	0	0	0	0	1
1	0	0	1	1	1	1	1	0	1
1	0	0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1	1	1

迷路のデータを定義する

二次元リストを使って、迷路を作る。まずは壁を作ってみよう。

```
import tkinter ・・・・tkinterモジュールの呼出し
root = tkinter.Tk() ・・・・ウィンドウのオブジェクトを作る
root.title("迷路の表示") ・・・・ウィンドウのタイトルを指定
canvas = tkinter.Canvas(width=800,height=560,bg="white")
                ・・・ キャンバスの部品（横:800,縦:560、背景色:ホワイト）を作る
canvas.pack() ・・・・キャンバスを配置する
m = [           ・・・・リストで迷路を定義する
    [1,1,1,1] ・・・・二次元リスト[ [0][0] , [0][1] , [0][2] , [0][3] ]
    ]
if m[0][0] ==1: ・・・・もし、m[y=0][x=0]が1、つまり壁なら灰色の四角を作る
    canvas.create_rectangle(0,0,80,80,fill="grey")
```

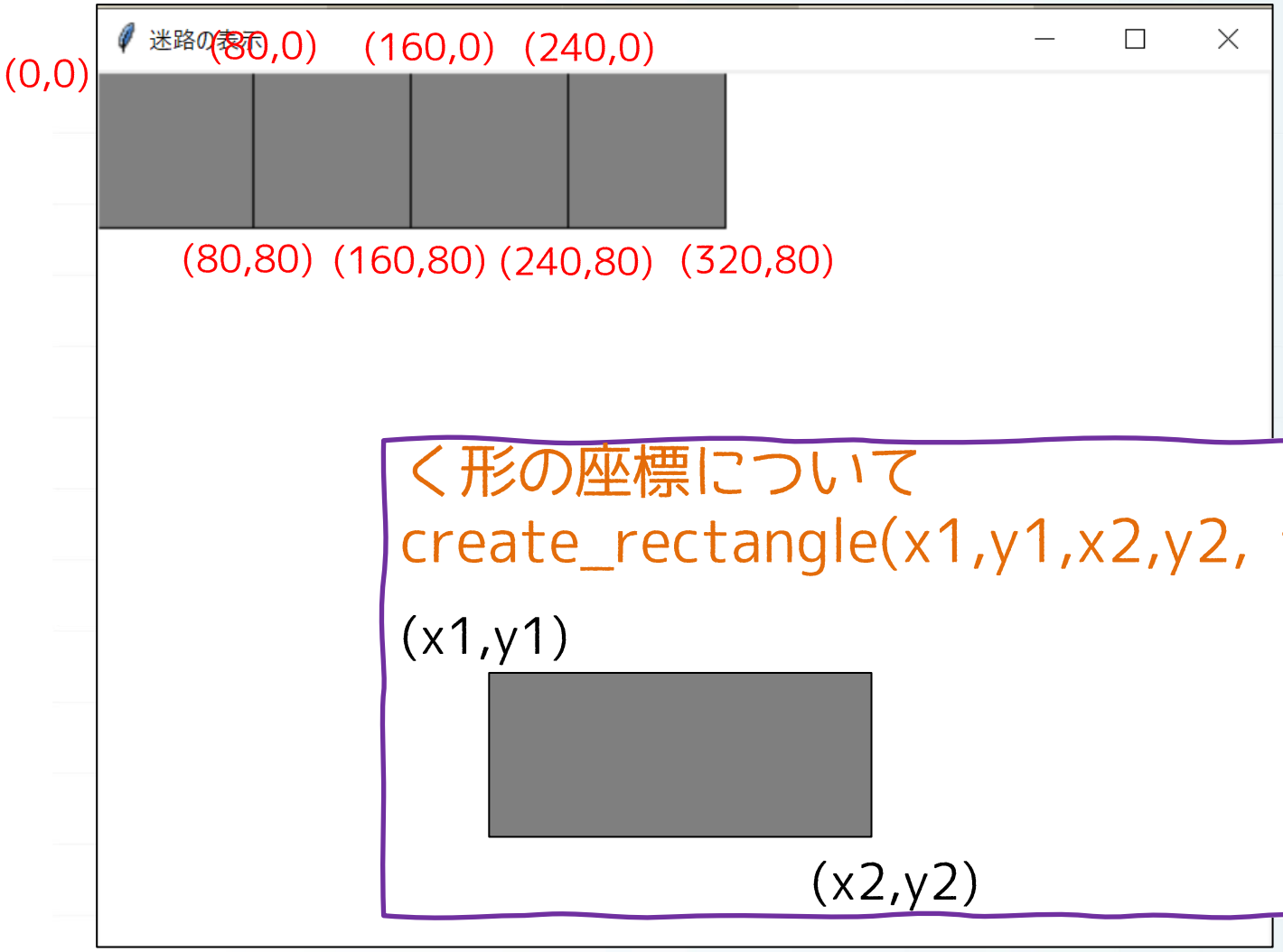
次のページに続く

迷路のデータを定義する

```
if m[0][1] ==1:  ・・・もし、m[y=0][x=1]が1、つまり壁なら灰色の四角を作る
    canvas.create_rectangle(80,0,160,80,fill="grey")
if m[0][2] ==1:  ・・・もし、m[y=0][x=2]が1、つまり壁なら灰色の四角を作る
    canvas.create_rectangle(160,0,240,80,fill="grey")
if m[0][3] ==1:  ・・・もし、m[y=0][x=3]が1、つまり壁なら灰色の四角を作る
    canvas.create_rectangle(240,0,320,80,fill="grey")
root.mainloop()  ・・・・ウィンドウを表示する
```

迷路のデータを定義する

「Run Module」またはF5キーを押してプログラムを実行する。



く形の座標について
create_rectangle(x1,y1,x2,y2, fill=塗りつぶす色)

迷路のデータを定義する

壁の次は二次元リストを使って、壁を囲んでみよう。

```
import tkinter      ・・・・tkinterモジュールの呼出し
root = tkinter.Tk()   ・・・・ウィンドウのオブジェクトを作る
root.title("迷路の表示")  ・・・・ウィンドウのタイトルを指定
canvas = tkinter.Canvas(width=800,height=560,bg="white")
                    ・・・ キャンバスの部品（横:800,縦:560、背景色:ホワイト）を作る
canvas.pack() ・・・・キャンバスを配置する
m = [               ・・・・リストで迷路を定義する
    [1,1,1,1], ・・・・二次元リスト[
    [1,0,0,1],                          [0][0] , [0][1] , [0][2] , [0][3]
    [1,1,1,1]                           [1][0] , [1][1] , [1][2] , [1][3]
                                        [2][0] , [2][1] , [2][2] , [2][3]
                                       ]
    ]
#1段目
```

次のページに続く

迷路のデータを定義する

```
if m[0][0] ==1: ・・・もし、m[y=0][x=0]が1なら
    canvas.create_rectangle(0,0,80,80,fill="grey")  ・・・灰色の四角を作る
if m[0][1] ==1: ・・・もし、m[y=0][x=1]が1なら
    canvas.create_rectangle(80,0,160,80,fill="grey") ・・・灰色の四角を作る
if m[0][2] ==1: ・・・もし、m[y=0][x=2]が1なら
    canvas.create_rectangle(160,0,240,80,fill="grey") ・・・灰色の四角を作る
if m[0][3] ==1: ・・・もし、m[y=0][x=3]が1なら
    canvas.create_rectangle(240,0,320,80,fill="grey") ・・・灰色の四角を作る
#2段目
if m[1][0] ==1:  ・・・もし、m[y=1][x=0]が1なら
    canvas.create_rectangle(0,80,80,160,fill="grey")  ・・・灰色の四角を作る
if m[1][1] ==1:  ・・・もし、m[y=1][x=1]が1なら
    canvas.create_rectangle(80,80,160,160,fill="grey")・・・灰色の四角を作る
```

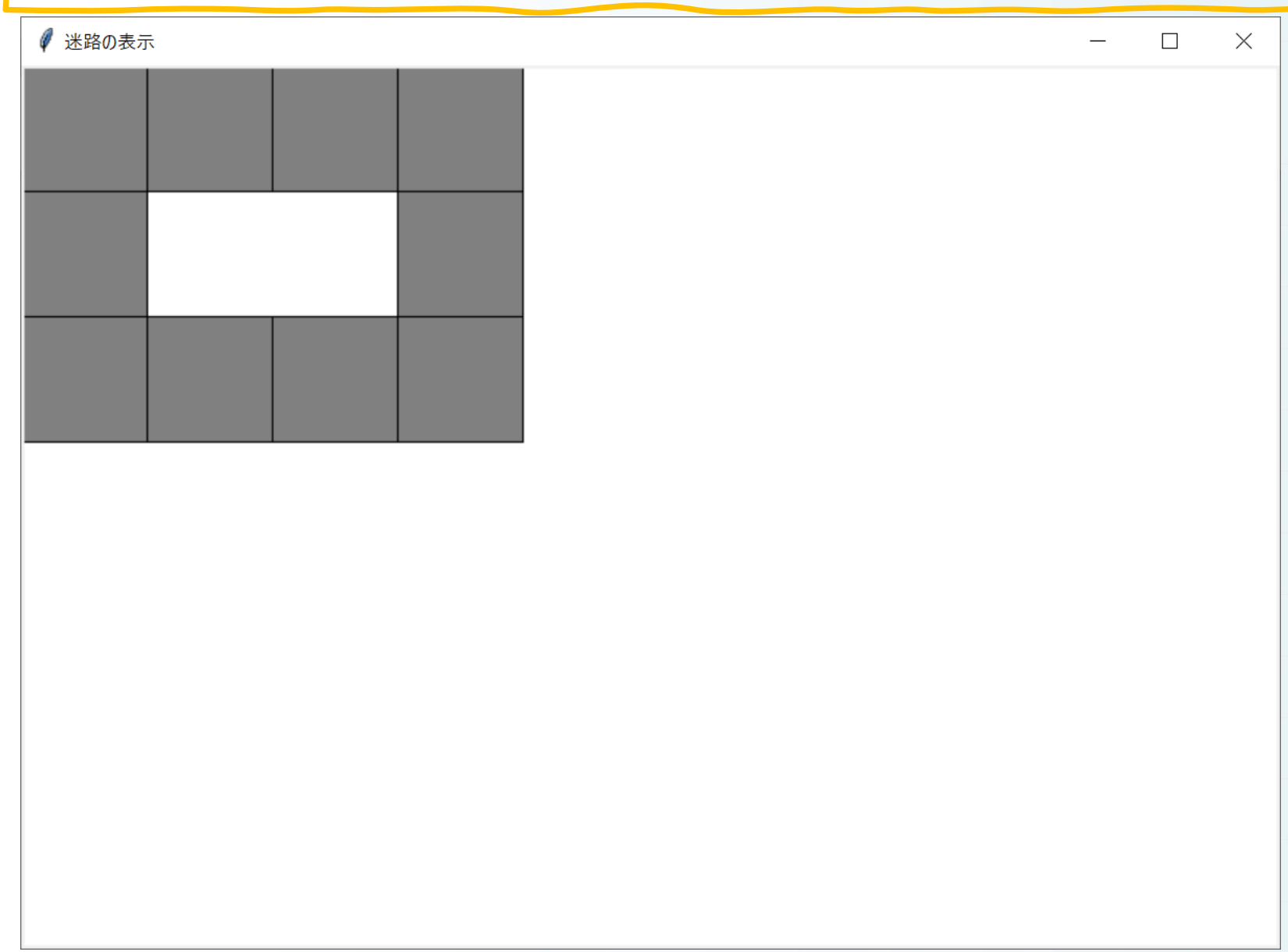
次のページに続く

迷路のデータを定義する

```
if m[1][2] ==1: ・・・もし、m[y=1][x=2]が1なら
    canvas.create_rectangle(80,160,160,240,fill="grey")・・灰色の四角を作る
if m[1][3] ==1: ・・・もし、m[y=1][x=3]が1なら
    canvas.create_rectangle(240,80,320,160,fill="grey") ・・灰色の四角を作る
#3段目
if m[2][0] ==1: ・・・もし、m[y=2][x=0]が1なら
    canvas.create_rectangle(0,160,80,240,fill="grey") ・・・灰色の四角を作る
if m[2][1] ==1: ・・・もし、m[y=2][x=1]が1なら
    canvas.create_rectangle(80,160,160,240,fill="grey") ・・灰色の四角を作る
if m[2][2] ==1: ・・・もし、m[y=2][x=2]が1なら
    canvas.create_rectangle(160,160,240,240,fill="grey")・・灰色の四角を作る
if m[2][3] ==1: ・・・もし、m[y=2][x=3]が1なら
    canvas.create_rectangle(240,160,320,240,fill="grey")・・灰色の四角を作る
root.mainloop()・・・・ウィンドウを表示する
```

迷路のデータを定義する

「Run Module」またはF5キーを押してプログラムを実行する。



迷路のデータを定義する

for文を使って、同じことをする。壁を囲むには二重ループのfor文を使用する。

```
for 変数1 in 変数1の範囲:
    for 変数2 in 変数2の範囲:
        処理のブロック
```

```
for y in range(3):
    for x in range(4):
```

xの値 0 ⇒ 1 ⇒ 2 ⇒ 3

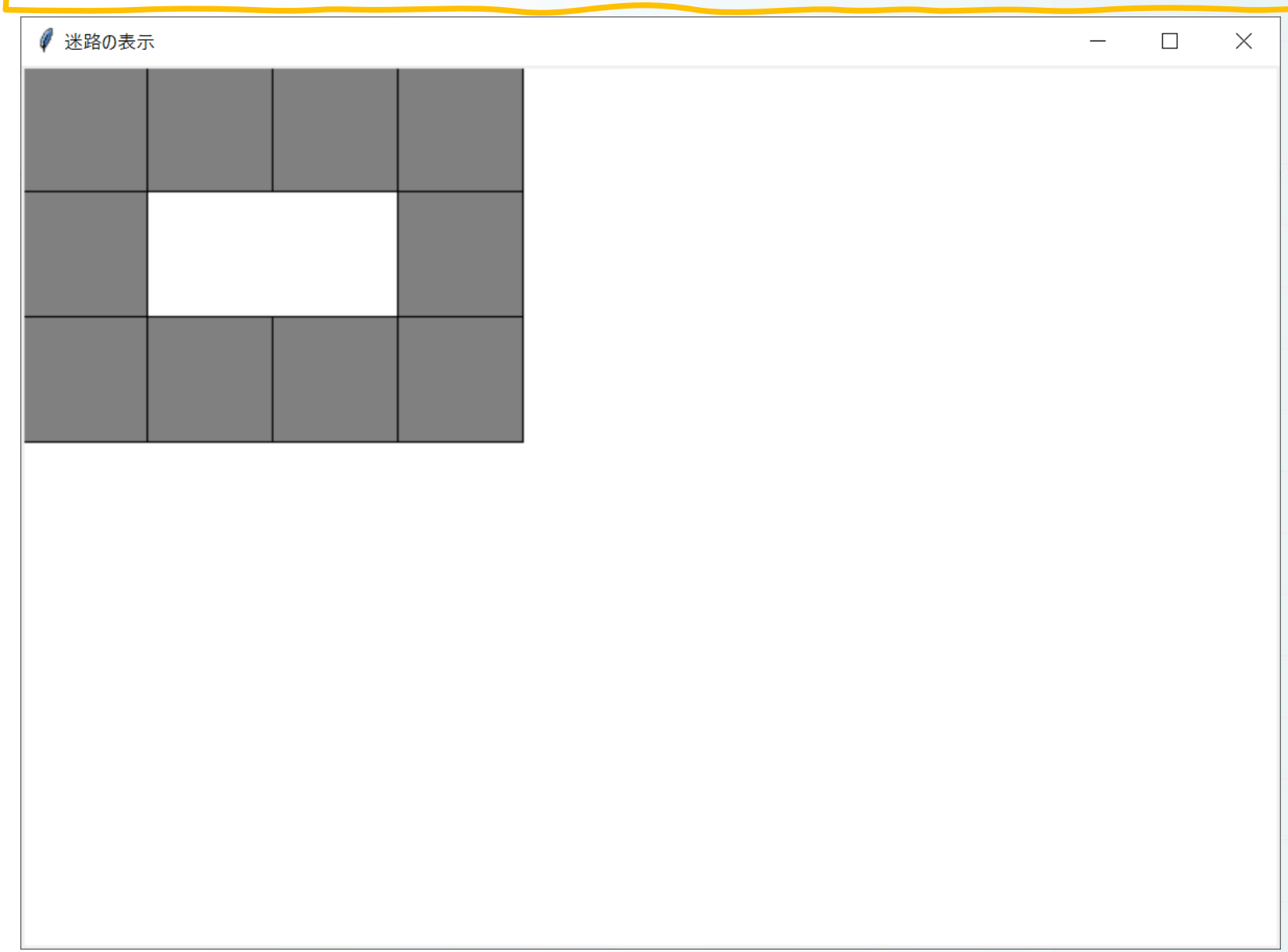
(y座標,x座標)

迷路のデータを定義する

```
import tkinter    ・・・・tkinterモジュールの呼出し
root = tkinter.Tk()   ・・・・ウィンドウのオブジェクトを作る
root.title("迷路の表示")    ・・・・ウィンドウのタイトルを指定
canvas = tkinter.Canvas(width=800,height=560,bg="white")
                  ・・・ キャンバスの部品（横:800,縦:560、背景色:ホワイト）を作る
canvas.pack()  ・・・・キャンバスを配置する
m = [    ・・・・リストで迷路を定義する
    [1,1,1,1],    ・・・・二次元リスト[
    [1,0,0,1],                          [0][0] , [0][1] , [0][2] , [0][3]
    [1,1,1,1]                           [1][0] , [1][1] , [1][2] , [1][3]
                                        [2][0] , [2][1] , [2][2] , [2][3]
    ]                                  ]
for y in range(3):  ・・・繰り返し yは 0 → 1 →2
    for x in range(4): ・・・繰り返し xは 0 → 1 → 2 → 3
        if m[y][x] == 1:  ・・・m[y][x]が1のとき
            canvas.create_rectangle(x*80,y*80,x*80+80,y*80+80,fill="gray")
root.mainloop()・・・・ウィンドウを表示する                    ・・・灰色の壁をつくる
```

迷路のデータを定義する

「Run Module」またはF5キーを押してプログラムを実行する。



迷路のデータを定義する

二重ループのforを使って迷路を作ってみよう。

```
import tkinter      ・・・・tkinterモジュールの呼出し
root = tkinter.Tk()   ・・・・ウィンドウのオブジェクトを作る
root.title("迷路の表示")  ・・・・ウィンドウのタイトルを指定
canvas = tkinter.Canvas(width=800,height=560,bg="white")
                    ・・・ キャンバスの部品（横:800,縦:560、背景色:ホワイト）を作る
canvas.pack()       ・・・・キャンバスを配置する
m = [               ・・・・リストで迷路を定義する
    [1,1,1,1,1,1,1,1,1,1],
    [1,0,0,0,0,0,1,0,0,1],
    [1,0,1,1,0,0,1,0,0,1],
    [1,0,0,1,0,0,0,0,0,1],
    [1,0,0,1,1,1,1,1,0,1],
    [1,0,0,0,0,0,0,0,0,1],
```

次のページに続く

迷路のデータを定義する

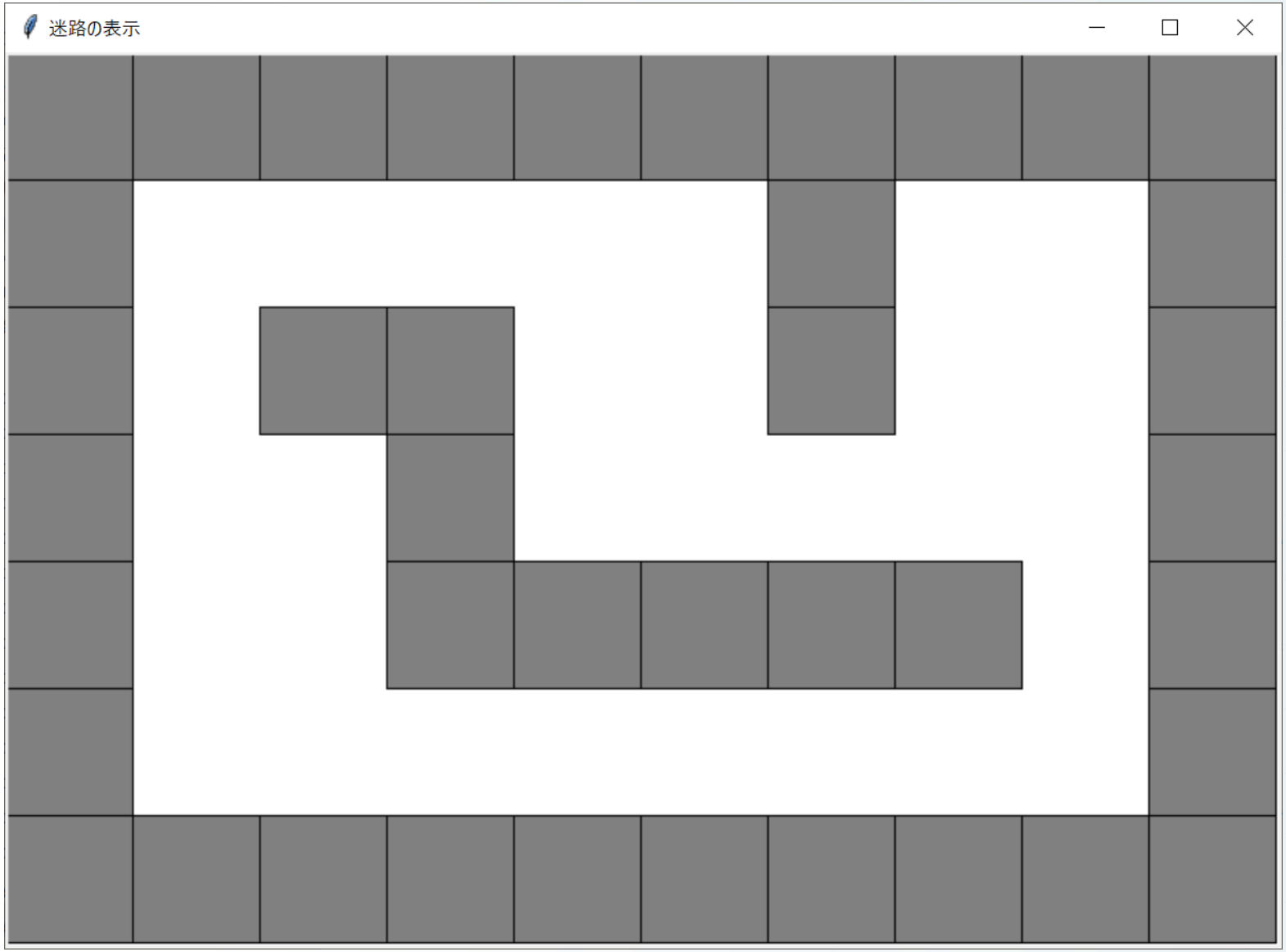
```
    [1,1,1,1,1,1,1,1,1,1]
    ]
for y in range(7):      ・・・繰り返し yは 0 → 1 →2
    for x in range(10): ・・・繰り返し xは 0 → 1 → 2 → 3
        if m[y][x] == 1:  ・・・m[y][x]が1のとき

canvas.create_rectangle(x*80,y*80,x*80+80,y*80+80,fill="gray")
                                                  ・・・灰色の壁をつくる

root.mainloop()  ・・・・ウィンドウを表示する
```

迷路のデータを定義する

「Run Module」またはF5キーを押してプログラムを実行する。



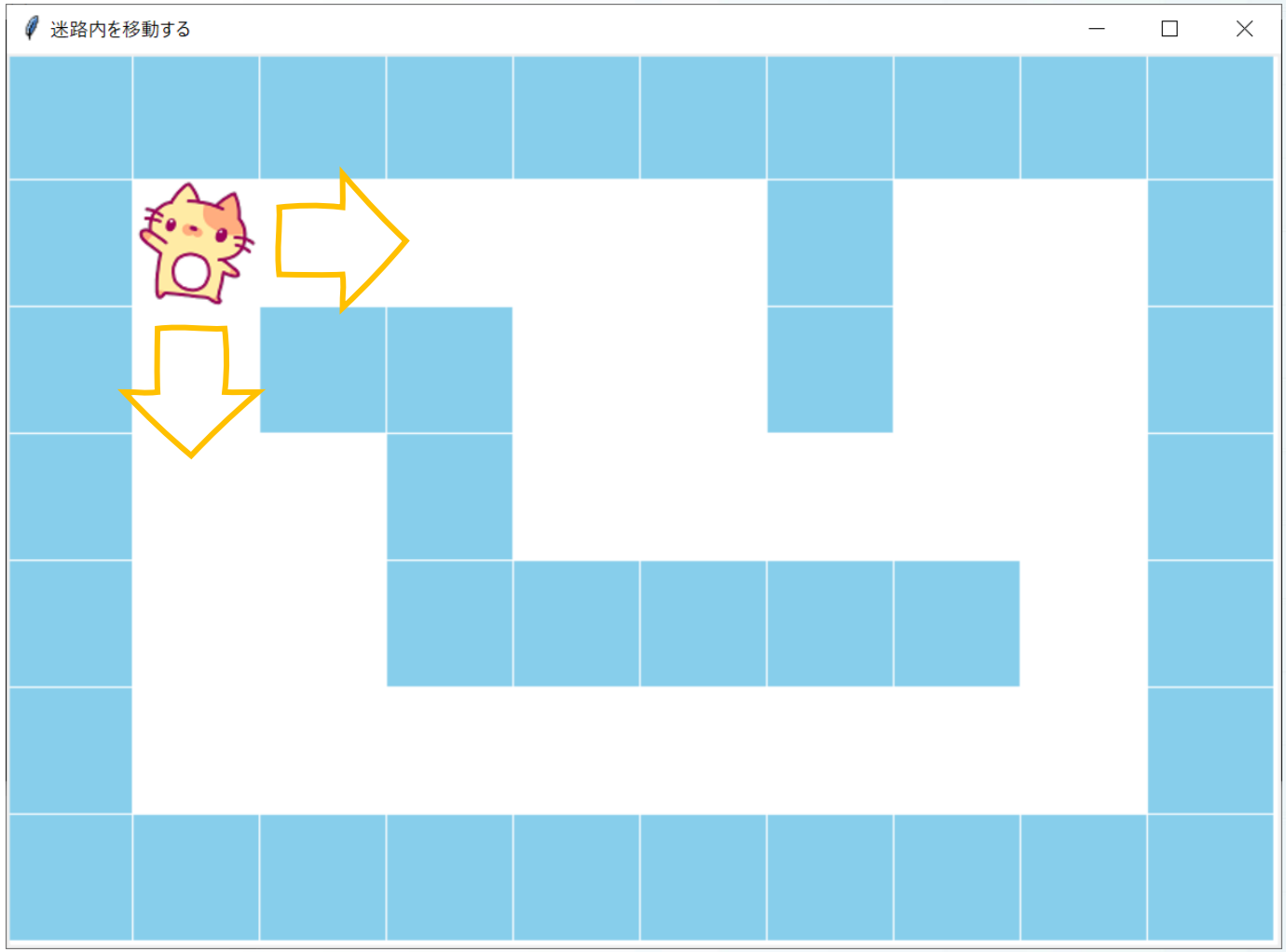
迷路のデータを定義する

【参考】二次元リストmaze[][]の添え字

maze [0][0]	maze [0][1]	maze [0][2]	maze [0][3]	maze [0][4]	maze [0][5]	maze [0][6]	maze [0][7]	maze [0][8]	maze [0][9]
maze [1][0]	maze [1][1]	maze [1][2]	maze [1][3]	maze [1][4]	maze [1][5]	maze [1][6]	maze [1][7]	maze [1][8]	maze [1][9]
maze [2][0]	maze [2][1]	maze [2][2]	maze [2][3]	maze [2][4]	maze [2][5]	maze [2][6]	maze [2][7]	maze [2][8]	maze [2][9]
maze [3][0]	maze [3][1]	maze [3][2]	maze [3][3]	maze [3][4]	maze [3][5]	maze [3][6]	maze [3][7]	maze [3][8]	maze [3][9]
maze [4][0]	maze [4][1]	maze [4][2]	maze [4][3]	maze [4][4]	maze [4][5]	maze [4][6]	maze [4][7]	maze [4][8]	maze [4][9]
maze [5][0]	maze [5][1]	maze [5][2]	maze [5][3]	maze [5][4]	maze [5][5]	maze [5][6]	maze [5][7]	maze [5][8]	maze [5][9]
maze [6][0]	maze [6][1]	maze [6][2]	maze [6][3]	maze [6][4]	maze [6][5]	maze [6][6]	maze [6][7]	maze [6][8]	maze [6][9]

二次元画面のゲーム開発の基礎

リアルタイム処理、キー入力、迷路の定義という３つの知識を使って、キャラクタを操作し迷路の中を歩けるプログラムを作る。



二次元画面のゲーム開発の基礎

```
import tkinter   ・・・・tkinterモジュールの呼出し

key =""          ・・・・変数「key」にブランクを代入

def key_down(e): ・・・ key_down関数を呼び出した時に実行する関数を定義
    global key   ・・・・keyをグローバル変数として扱うと宣言
    key= e.keysym   ・・・ 押されたキーの名称を変数「key」に代入
def key_up(e):       ・・・・キーを離した時に実行する関数の定義
    global key       ・・・・keyをグローバル変数として扱うと宣言
    key =""          ・・・・keyに空の文字列を代入

mx=1             ・・・・キャラクタの横方向を管理する変数
my=1             ・・・・キャラクタの縦方向を管理する変数

def main_proc():  ・・・・リアルタイム処理を行う関数を定義
    global mx,my  ・・・・mx,myをグローバル変数として扱うと宣言
```

次のページに続く

二次元画面のゲーム開発の基礎

```
    if key == "Up" and maze[my-1][mx]==0:
        my = my - 1  ・・・方向キーの上が押され、かつ、上のマスが通路ならmyを1減らす
    if key == "Down" and maze[my+1][mx]==0:
        my = my + 1  ・・・方向キーの下が押され、かつ、下のマスが通路ならmyを1増やす
    if key =="Left" and maze[my][mx-1]==0:
        mx = mx - 1  ・・・方向キーの左が押され、かつ、左のマスが通路ならmxを1減らす
    if key == "Right" and maze[my][mx+1]==0:
        mx = mx + 1  ・・・方向キーの右が押され、かつ、右のマスが通路ならmxを1増やす
    canvas.coords("MYCHR",mx*80+40,my*80+40)
                          ・・・・キャラクタ(タグ：MYCHR)を新しい座標に移動する
    root.after(100,main_proc)
                          ・・・after()命令で0.1秒後にmain_proc()関数を実行する
root=tkinter.Tk()         ・・・・ウィンドウのオブジェクトを作る
root.title("迷路内を移動する") ・・・・ウィンドウのタイトルを指定
```

次のページに続く

二次元画面のゲーム開発の基礎

```
root.bind("<KeyPress>",key_down)
```
・・・ bind()命令でキーを押した時に実行する関数を指定する
```
root.bind("<KeyRelease>",key_up)
```
・・・ bind()命令でキーを離した時に実行する関数を指定する
```
canvas = tkinter.Canvas(width=800,height=560,bg="white")
```
・・・ キャンバスの部品（横:800,縦:560、背景色:白）を作る
```
canvas.pack()
```
・・・・キャンバスを配置する
```
maze = [
    [1,1,1,1,1,1,1,1,1,1],
    [1,0,0,0,0,0,1,0,0,1],
    [1,0,1,1,0,0,1,0,0,1],
    [1,0,0,1,0,0,0,0,0,1],
    [1,0,0,1,1,1,1,1,0,1],
    [1,0,0,0,0,0,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1]
    ]
```
・・・・リストで迷路を定義する

次のページに続く

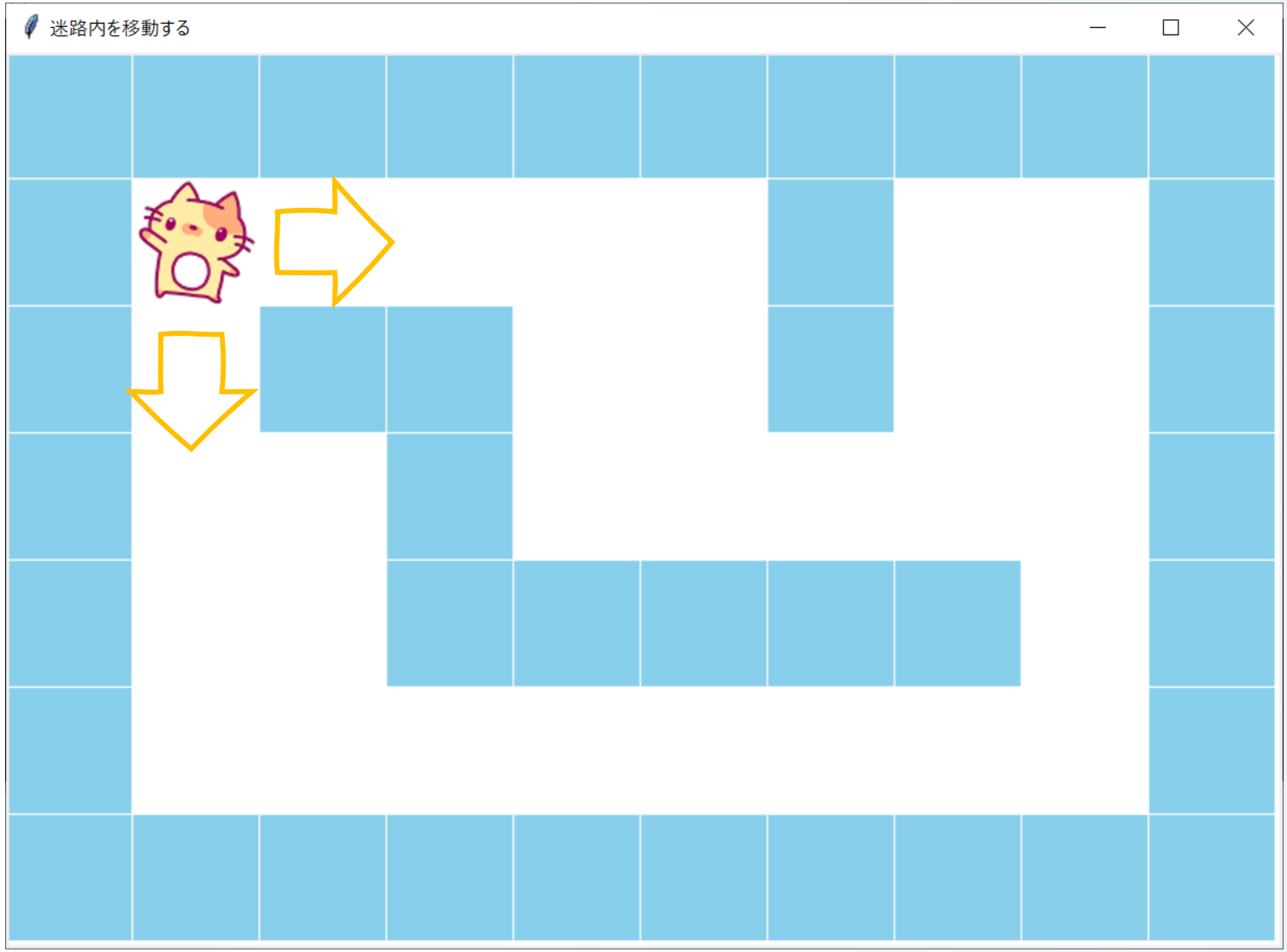
二次元画面のゲーム開発の基礎

```
for y in range(7):  ・・・繰り返し yは 0 → 1 → 2 → 3 → 4 → 5 → 6
    for x in range(10):  ・・・繰り返し xは 0 → 1 → 2 → 3 → 4 → ……
        if maze[y][x]==1:  ・・・m[y][x]が1のとき
            canvas.create_rectangle(x*80,y*80,x*80+79,y*80+79,fill =
"skyblue",width=0)  ・・・スカイブルーの四角をつくる

img = tkinter.PhotoImage(file="mimi_s.png")
                                        ・・キャラクタの画像をimgに読み込む
canvas.create_image(mx*80+40,my*80+40,image=img,tag="MYCHR")
                                        ・・キャンバスに画像を表示
main_proc()  ・・・main_proc()関数を実行する
root.mainloop()  ・・・・ウィンドウを表示する
```

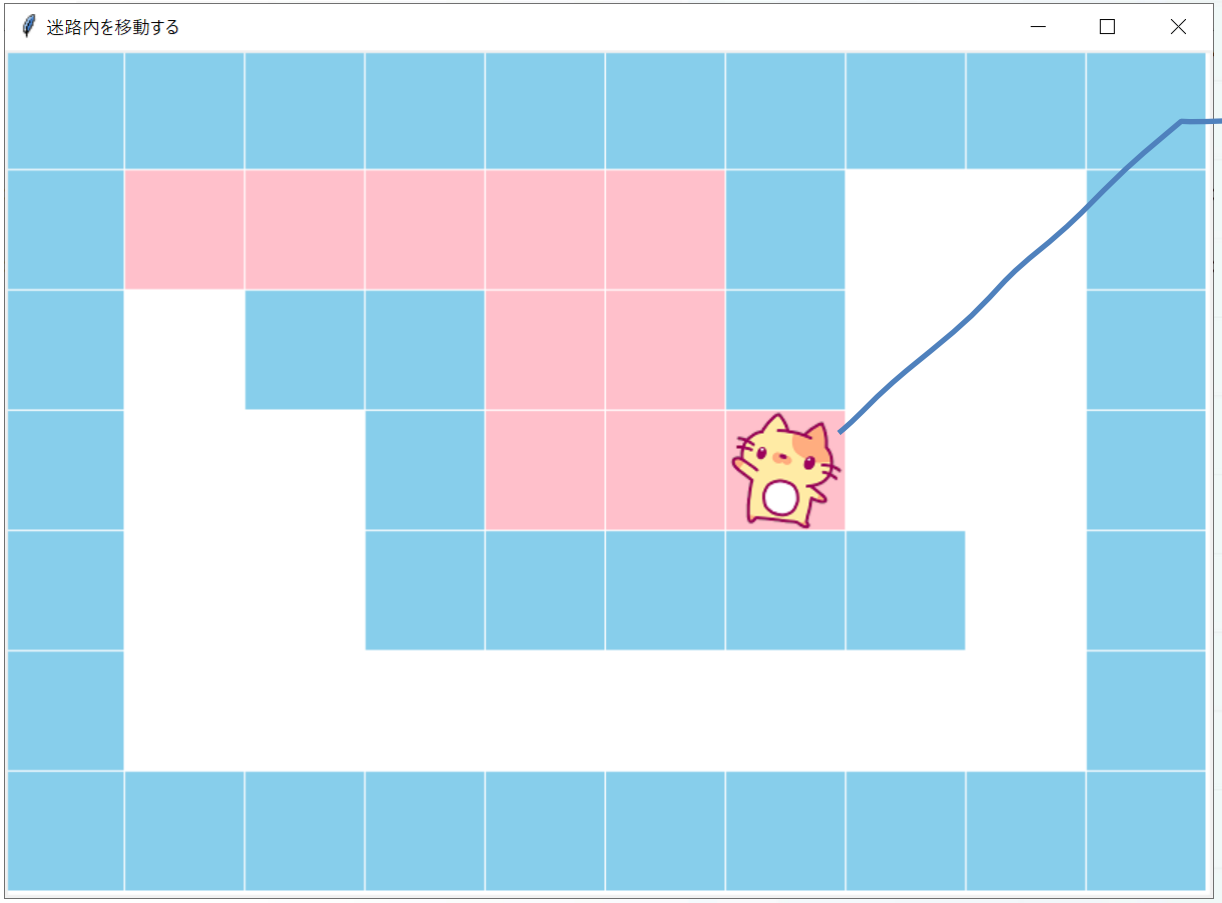
二次元画面のゲーム開発の基礎

「Run Module」またはF5キーを押してプログラムを実行する。



ゲームとして完成させる

リストの値を書き換えて、キャラクタが通った所をピンク色で塗ってみよう。二次元リストの迷路のデータの値は通路が0,壁が1となっている。通った所は0から２にすると一筆書きのルールができる。



一度、通ったところはピンク色になり、戻ることができない。

ゲームとして完成させる

```
import tkinter   ・・・・tkinterモジュールの呼出し

key =""          ・・・・変数「key」にブランクを代入
def key_down(e): ・・・ key_down関数を呼び出した時に実行する関数を定義
    global key   ・・・・keyをグローバル変数として扱うと宣言
    key= e.keysym   ・・・ 押されたキーの名称を変数「key」に代入
def key_up(e):       ・・・・キーを離した時に実行する関数の定義
    global key       ・・・・keyをグローバル変数として扱うと宣言
    key =""          ・・・・keyに空の文字列を代入

mx=1             ・・・・キャラクタの横方向を管理する変数
my=1             ・・・・キャラクタの縦方向を管理する変数

def main_proc():  ・・・・リアルタイム処理を行う関数を定義
    global mx,my  ・・・・mx,myをグローバル変数として扱うと宣言
```

次のページに続く

ゲームとして完成させる

```
if key == "Up" and maze[my-1][mx]==0:
    my = my - 1  ・・・方向キーの上が押され、かつ、上のマスが通路ならmyを1減らす
if key == "Down" and maze[my+1][mx]==0:
    my = my + 1  ・・・方向キーの下が押され、かつ、下のマスが通路ならmyを1増やす
if key =="Left" and maze[my][mx-1]==0:
    mx = mx - 1  ・・・方向キーの左が押され、かつ、左のマスが通路ならmxを1減らす
if key == "Right" and maze[my][mx+1]==0:
    mx = mx + 1  ・・・方向キーの右が押され、かつ、右のマスが通路ならmxを1増やす
if maze[my][mx] ==0:
    maze[my][mx]=2・・・・キャラクタ(タグ：MYCHR)を新しい座標に移動する
    canvas.create_rectangle(mx*80,my*80,mx*80+79,my*80+79,
    fill="pink",width=0) ・・・・キャラクタのいる場所が通路ならリストの値を
                              2にし、そこをピンク色で塗る
canvas.delete("MYCHR")  ・・・一旦、キャラクタを消す
```

次のページに続く

ゲームとして完成させる

```
    canvas.create_image(mx*80+40,my*80+40,image=img,tag="MYCHR")
                                                    ・・・消したキャラクタを再び表示する
    root.after(100,main_proc)
                                ・・・after()命令で0.1秒後にmain_proc()関数を実行する
root=tkinter.Tk()          ・・・・ウィンドウのオブジェクトを作る
root.title("迷路内を移動する")・・・・ウィンドウのタイトルを指定
root.bind("<KeyPress>",key_down)
                                    ・・・ bind()命令でキーを押した時に実行する関数を指定する
root.bind("<KeyRelease>",key_up)
                                    ・・・ bind()命令でキーを離した時に実行する関数を指定する
canvas = tkinter.Canvas(width=800,height=560,bg="white")
                                ・・・ キャンバスの部品（横:800,縦:560、背景色:白）を作る
canvas.pack() ・・・・キャンバスを配置する
maze = [                                    ・・・・リストで迷路を定義する
    [1,1,1,1,1,1,1,1,1,1],
    [1,0,0,0,0,0,1,0,0,1],
```

次のページに続く

ゲームとして完成させる

```
    [1,0,1,1,0,0,1,0,0,1],
    [1,0,0,1,0,0,0,0,0,1],
    [1,0,0,1,1,1,1,1,0,1],
    [1,0,0,0,0,0,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1]
    ]


for y in range(7):      ・・・繰り返し yは 0 → 1 → 2 → 3 → 4 → 5 → 6
    for x in range(10):     ・・・繰り返し xは 0 → 1 → 2 → 3 → 4 → ……
        if maze[y][x]==1:     ・・・m[y][x]が1のとき
            canvas.create_rectangle(x*80,y*80,x*80+79,y*80+79,fill =
            "skyblue",width=0)   ・・・スカイブルーの四角をつくる
img = tkinter.PhotoImage(file="mimi_s.png")・・キャラクタの画像をimgに読み込む
```

次のページに続く

ゲームとして完成させる

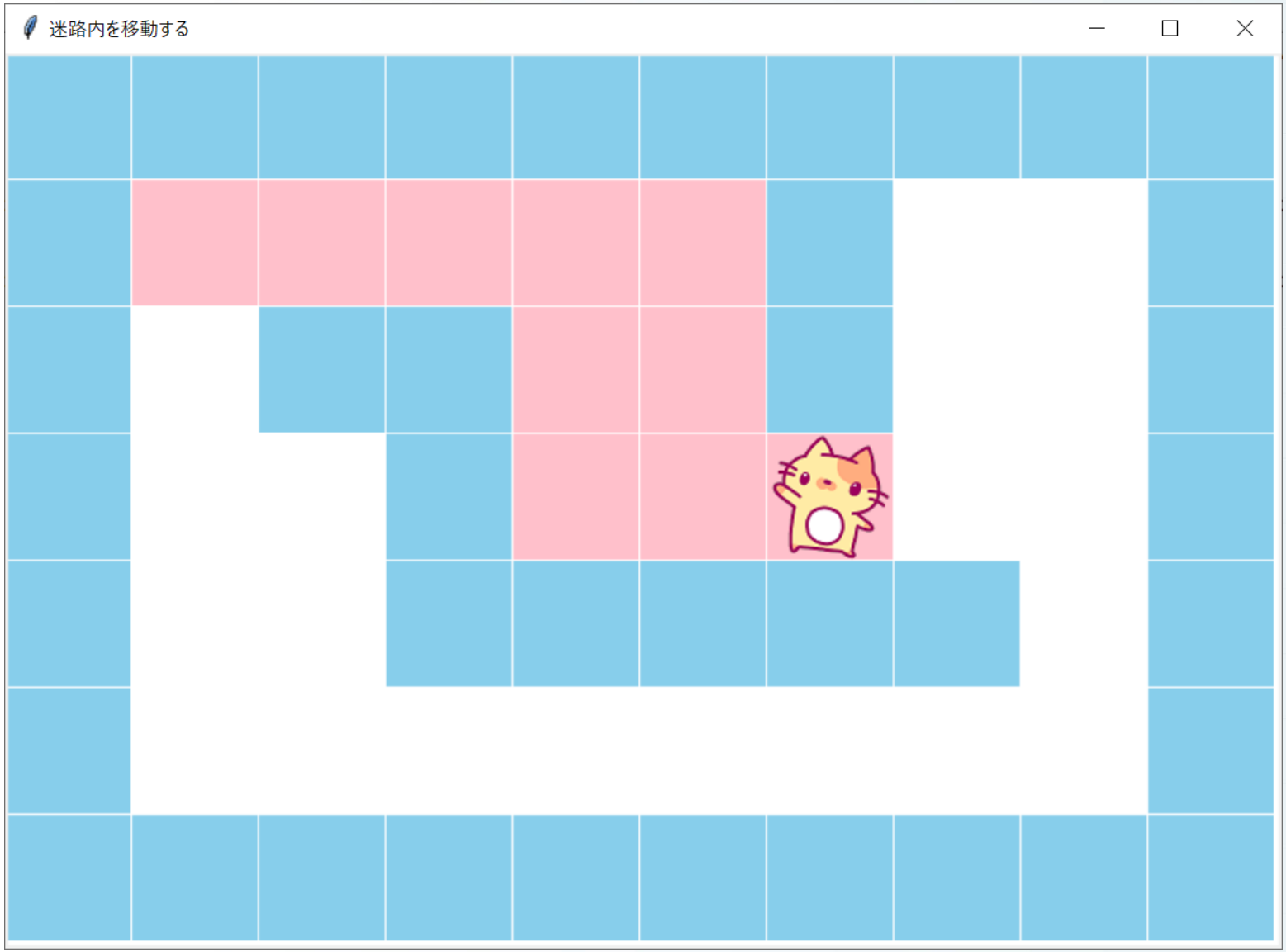
```
canvas.create_image(mx*80+40,my*80+40,image=img,tag="MYCHR")
```
・・・・キャンパスに画像を表示

```
main_proc()
```
・・・main_proc()関数を実行する

```
root.mainloop()
```
・・・・ウィンドウを表示する

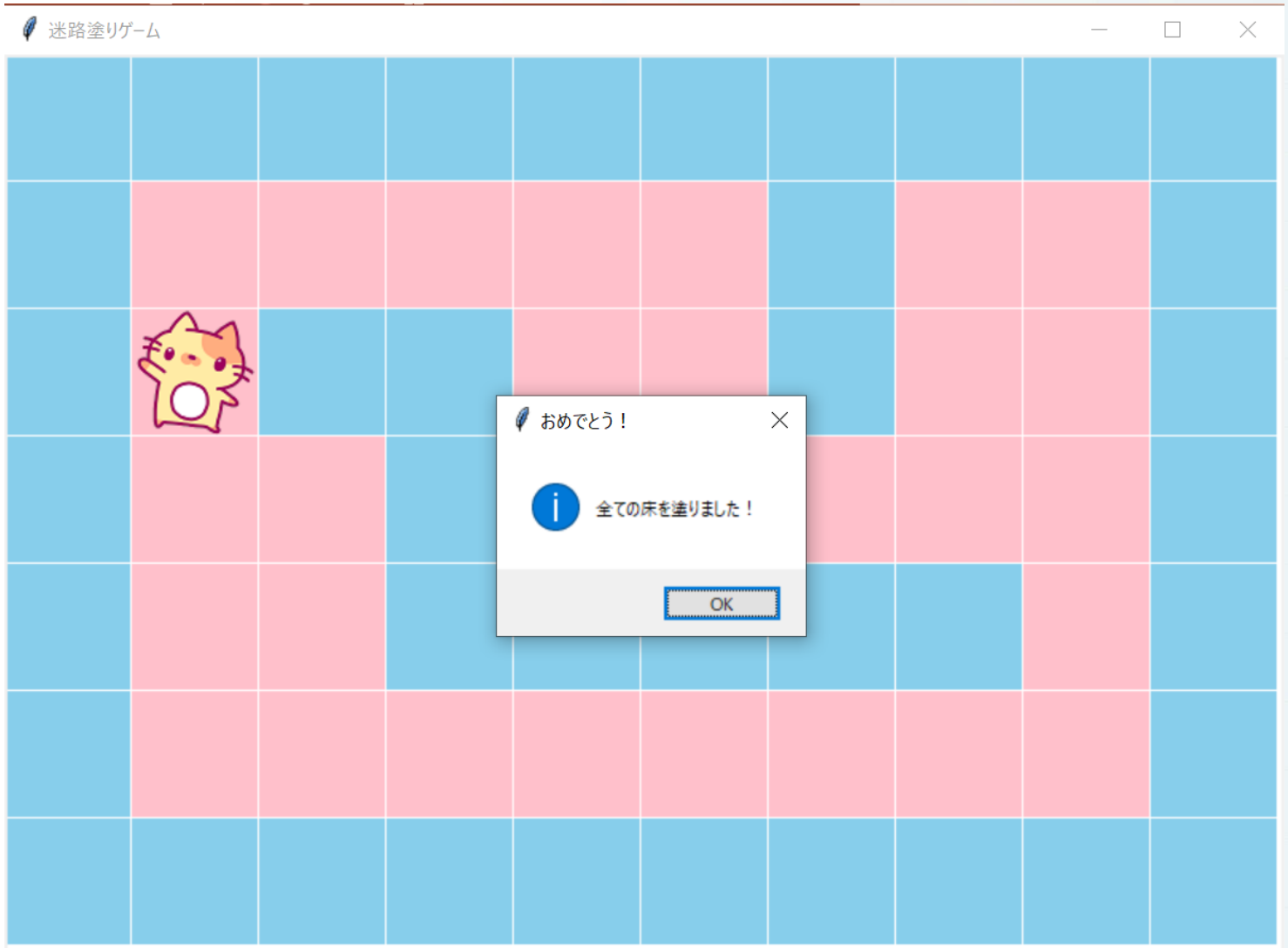
ゲームとして完成させる

「Run Module」またはF5キーを押してプログラムを実行する。



ゲームとして完成させる

全ての床を塗ったか判定する処理を追加する。判定はif文で行う。



ゲームとして完成させる

```
import tkinter   ・・・・tkinterモジュールの呼出し
import tkinter.messagebox ・・・・tkinter.messageboxモジュールの呼出し
key =""          ・・・・変数「key」にブランクを代入
def key_down(e):  ・・・ key_down関数を呼び出した時に実行する関数を定義
    global key     ・・・・keyをグローバル変数として扱うと宣言
    key= e.keysym   ・・・ 押されたキーの名称を変数「key」に代入
def key_up(e):       ・・・・キーを離した時に実行する関数の定義
    global key       ・・・・keyをグローバル変数として扱うと宣言
    key =""          ・・・・keyに空の文字列を代入
mx=1                 ・・・・キャラクタの横方向を管理する変数
my=1                 ・・・・キャラクタの縦方向を管理する変数
yuka=0               ・・・・塗った床を数える変数、「yuka」に0を代入
def main_proc():     ・・・・リアルタイム処理を行う関数を定義
```

次のページに続く

ゲームとして完成させる

```
global mx,my,yuka ・・・・mx,my,yukaをグローバル変数として扱うと宣言
if key == "Up" and maze[my-1][mx]==0:
    my = my - 1          ・・・方向キーの上が押され、かつ、上のマスが通路なら
                          myを1減らす
if key == "Down" and maze[my+1][mx]==0:
    my = my + 1          ・・・方向キーの下が押され、かつ、下のマスが通路なら
                          myを1増やす
if key =="Left" and maze[my][mx-1]==0:
    mx = mx - 1          ・・・方向キーの左が押され、かつ、左のマスが通路なら
                          mxを1減らす
if key == "Right" and maze[my][mx+1]==0:
    mx = mx + 1          ・・・方向キーの右が押され、かつ、右のマスが通路なら
                          mxを1増やす
if maze[my][mx] == 0:    ・・・・キャラクタ(タグ：MYCHR)を新しい
                              座標に移動する
    maze[my][mx]= 2
    yuka = yuka + 1      ・・・・キャラクタのいる場所が通路ならリスト
                          の値を2にし、そこをピンク色で塗る
    canvas.create_rectangle(mx*80,my*80,mx*80+79,my*80+79,
                            fill="pink",width=0)
```

次のページに続く

ゲームとして完成させる

```
    canvas.delete("MYCHR")  ・・・一旦、キャラクタを消す
    canvas.create_image(mx*80+40,my*80+40,image=img,
                        tag="MYCHR")・・消したキャラクタを再び表示する
    if yuka == 30:      ・・・・もし、30箇所の床を塗ったら
                              キャンバスを更新する
        canvas.update()
        tkinter.messagebox.showinfo("おめでとう!","全ての床を塗りまし
                                    た!") ・・クリアメッセージを表示する
    else:
        root.after(200,main_proc)
                    ・・・after()命令で0.2秒後にmain_proc()関数を実行する

root=tkinter.Tk()       ・・・・ウィンドウのオブジェクトを作る
root.title("床塗りゲーム")  ・・・・ウィンドウのタイトルを指定
root.bind("<KeyPress>",key_down)
                    ・・・ bind()命令でキーを押した時に実行する関数を指定する
```

次のページに続く

ゲームとして完成させる

```
root.bind("<KeyRelease>",key_up)
```
・・・ bind()命令でキーを離した時に実行する関数を指定する

```
canvas = tkinter.Canvas(width=800,height=560,bg="white")
```
・・・ キャンバスの部品（横:800,縦:560、背景色:白）を作る

```
canvas.pack()
```
・・・・キャンバスを配置する

```
maze = [
    [1,1,1,1,1,1,1,1,1,1],
    [1,0,0,0,0,0,1,0,0,1],
    [1,0,1,1,0,0,1,0,0,1],
    [1,0,0,1,0,0,0,0,0,1],
    [1,0,0,1,1,1,1,1,0,1],
    [1,0,0,0,0,0,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1]
    ]
```
・・・・リストで迷路を定義する

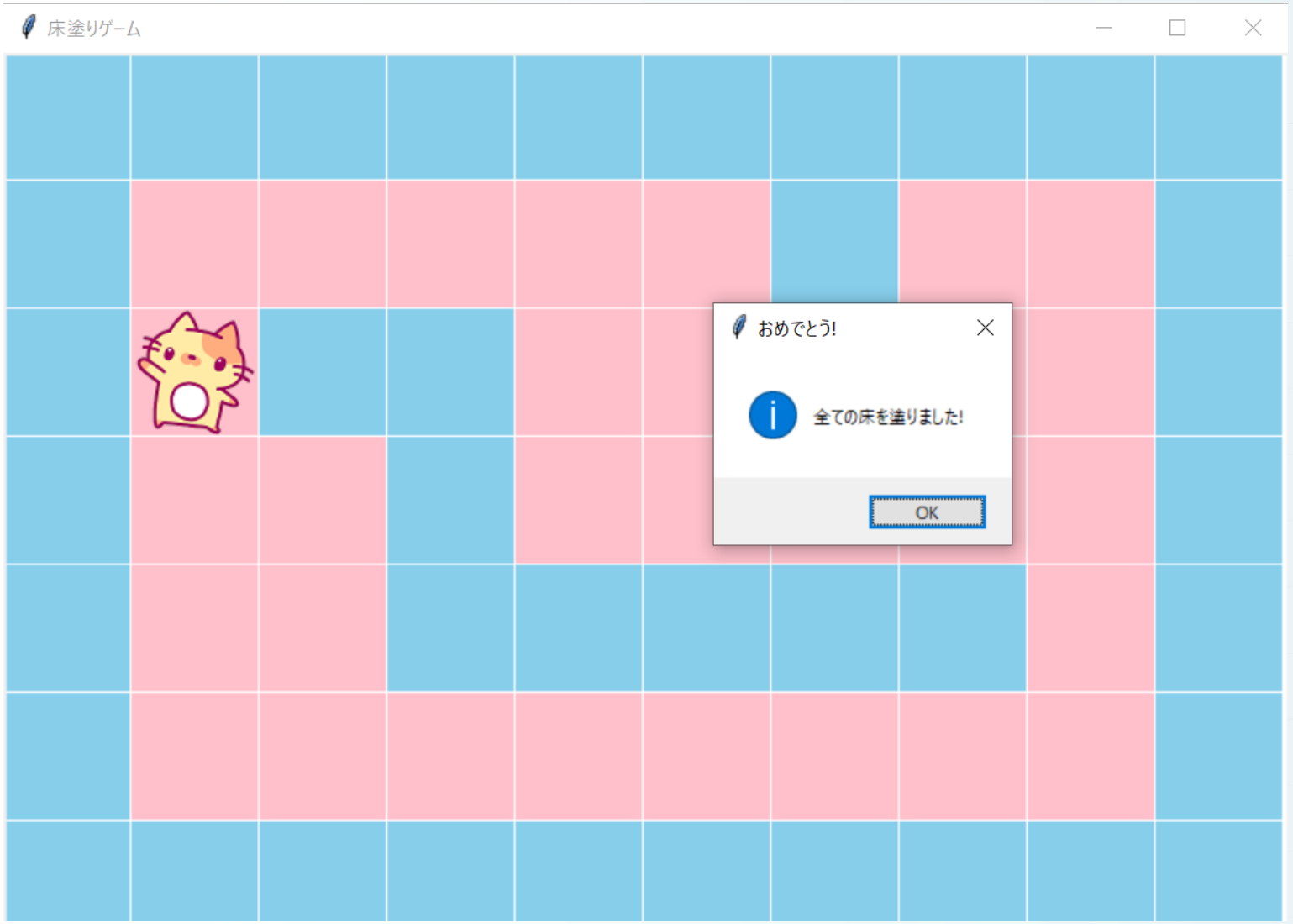
次のページに続く

ゲームとして完成させる

```
for y in range(7):  ・・・繰り返し yは 0 → 1 → 2 → 3 → 4 → 5 → 6
    for x in range(10):  ・・・繰り返し xは 0 → 1 → 2 → 3 → 4 → ……
        if maze[y][x]==1:  ・・・m[y][x]が1のとき
            canvas.create_rectangle(x*80,y*80,x*80+79,y*80+79,fill
= "skyblue",width=0)  ・・・スカイブルーの四角をつくる
img = tkinter.PhotoImage(file="mimi_s.png")
                                        ・・キャラクタの画像をimgに読み込む
canvas.create_image(mx*80+40,my*80+40,image=img,tag="MYCHR")
                                        ・・・・キャンバスに画像を表示
main_proc()  ・・・main_proc()関数を実行する
root.mainloop()・・・・ウィンドウを表示する
```

ゲームとして完成させる

「Run Module」またはF5キーを押してプログラムを実行する。



やり直せる処理を追加する

塗り方を間違えてしまった時に、やり直せる処理を追加しよう。
左の「Shift」キーを押すと最初からやり直せるように改良する。

```
import tkinter   ・・・・tkinterモジュールの呼出し
import tkinter.messagebox ・・・・tkinter.messageboxモジュールの呼出し
key =""          ・・・・変数「key」にブランクを代入
def key_down(e):  ・・・ key_down関数を呼び出した時に実行する関数を定義
    global key     ・・・・keyをグローバル変数として扱うと宣言
    key= e.keysym    ・・・ 押されたキーの名称を変数「key」に代入
def key_up(e):        ・・・・キーを離した時に実行する関数の定義
    global key        ・・・・keyをグローバル変数として扱うと宣言
    key =""           ・・・・keyに空の文字列を代入
mx=1                  ・・・・キャラクタの横方向を管理する変数
my=1                  ・・・・キャラクタの縦方向を管理する変数
```

次のページに続く

やり直せる処理を追加する

```
yuka=0 ・・・・塗った床を数える変数、「yuka」に0を代入
def main_proc():・・・・リアルタイム処理を行う関数を定義
    global mx,my,yuka ・・・・mx,my,yukaをグローバル変数として扱うと宣言
    if key == "Shift_L" and yuka > 1: ・・・・左Shiftキーを押し、かつ、2マス以上塗っていたら塗った床を消す
        canvas.delete("PAINT")
        mx=1 ・・・・mxに1代入
        my=1 ・・・・myに1代入
        yuka=0・・・・yukaに0を代入
        for y in range(7): ・・・・二重ループの繰り返し、外側のfor
            for x in range(10): ・・・・内側のfor
                if maze[y][x] ==2: ・・・・塗った床があれば
                    maze[y][x] =0  ・・・・値を0(塗っていない状態)に
```

キャラクタを初期位置に戻す

次のページに続く

やり直せる処理を追加する

```
if key == "Up" and maze[my-1][mx]==0:
    my = my - 1
if key == "Down" and maze[my+1][mx]==0:
    my = my + 1
if key =="Left" and maze[my][mx-1]==0:
    mx = mx - 1
if key == "Right" and maze[my][mx+1]==0:
    mx = mx + 1
if maze[my][mx] == 0:
    maze[my][mx]= 2
    yuka = yuka + 1
    canvas.create_rectangle(mx*80,my*80,mx*80+79,my*80+79,
                            fill="pink",width=0,tag=" PAINT" )
```

・・・方向キーの上が押され、かつ、上のマスが通路ならmyを1減らす

・・・方向キーの下が押され、かつ、下のマスが通路ならmyを1増やす

・・・方向キーの左が押され、かつ、左のマスが通路ならmxを1減らす

・・・方向キーの右が押され、かつ、右のマスが通路ならmxを1増やす

・・・・キャラクタ(タグ：MYCHR)を新しい座標に移動する

・・・・キャラクタのいる場所が通路ならリストの値を2にし、そこをピンク色で塗る

次のページに続く

やり直せる処理を追加する

```
    canvas.delete("MYCHR")  ・・・一旦、キャラクタを消す
    canvas.create_image(mx*80+40,my*80+40,image=img,
                        tag="MYCHR")・・消したキャラクタを再び表示する
    if yuka == 30:      ・・・・もし、30箇所の床を塗ったら
                                キャンバスを更新する
        canvas.update()
        tkinter.messagebox.showinfo("おめでとう!","全ての床を塗りまし
                                                    た!") ・・クリアメッセージを表示する
    else:
        root.after(200,main_proc)
                    ・・・after()命令で0.2秒後にmain_proc()関数を実行する

root=tkinter.Tk()       ・・・・ウィンドウのオブジェクトを作る
root.title("床塗りゲーム")  ・・・・ウィンドウのタイトルを指定
root.bind("<KeyPress>",key_down)
                    ・・・ bind()命令でキーを押した時に実行する関数を指定する
```

次のページに続く

やり直せる処理を追加する

```
root.bind("<KeyRelease>",key_up)
```
・・・ bind()命令でキーを離した時に実行する関数を指定する

```
canvas = tkinter.Canvas(width=800,height=560,bg="white")
```
・・・ キャンバスの部品（横:800,縦:560、背景色:白）を作る

```
canvas.pack()
```
・・・・キャンバスを配置する

```
maze = [
    [1,1,1,1,1,1,1,1,1,1],
    [1,0,0,0,0,0,1,0,0,1],
    [1,0,1,1,0,0,1,0,0,1],
    [1,0,0,1,0,0,0,0,0,1],
    [1,0,0,1,1,1,1,1,0,1],
    [1,0,0,0,0,0,0,0,0,1],
    [1,1,1,1,1,1,1,1,1,1]
    ]
```
・・・・リストで迷路を定義する

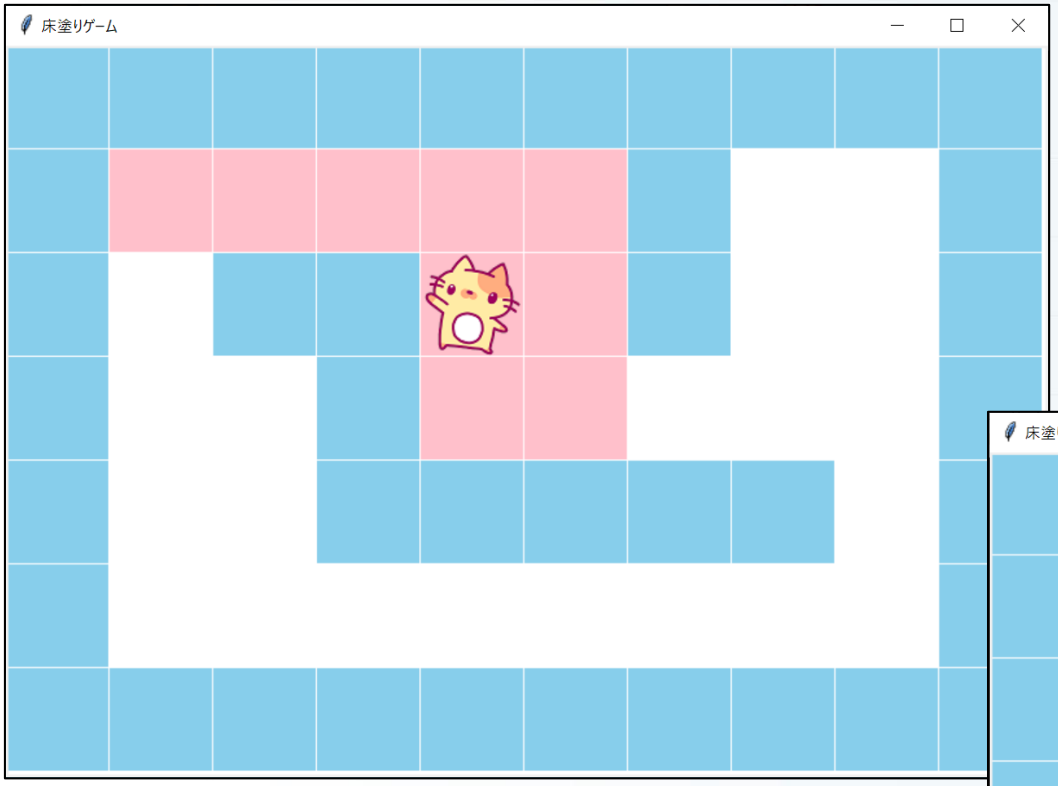
次のページに続く

やり直せる処理を追加する

```
for y in range(7):  ・・・繰り返し yは 0 → 1 → 2 → 3 → 4 → 5 → 6
    for x in range(10):  ・・・繰り返し xは 0 → 1 → 2 → 3 → 4 → ……
        if maze[y][x]==1:  ・・・m[y][x]が1のとき
            canvas.create_rectangle(x*80,y*80,x*80+79,y*80+79,fill
= "skyblue",width=0)  ・・・スカイブルーの四角をつくる
img = tkinter.PhotoImage(file="mimi_s.png")
                                        ・・キャラクタの画像をimgに読み込む
canvas.create_image(mx*80+40,my*80+40,image=img,tag="MYCHR")
                                        ・・・・キャンバスに画像を表示
main_proc()  ・・・main_proc()関数を実行する
root.mainloop()・・・・ウィンドウを表示する
```

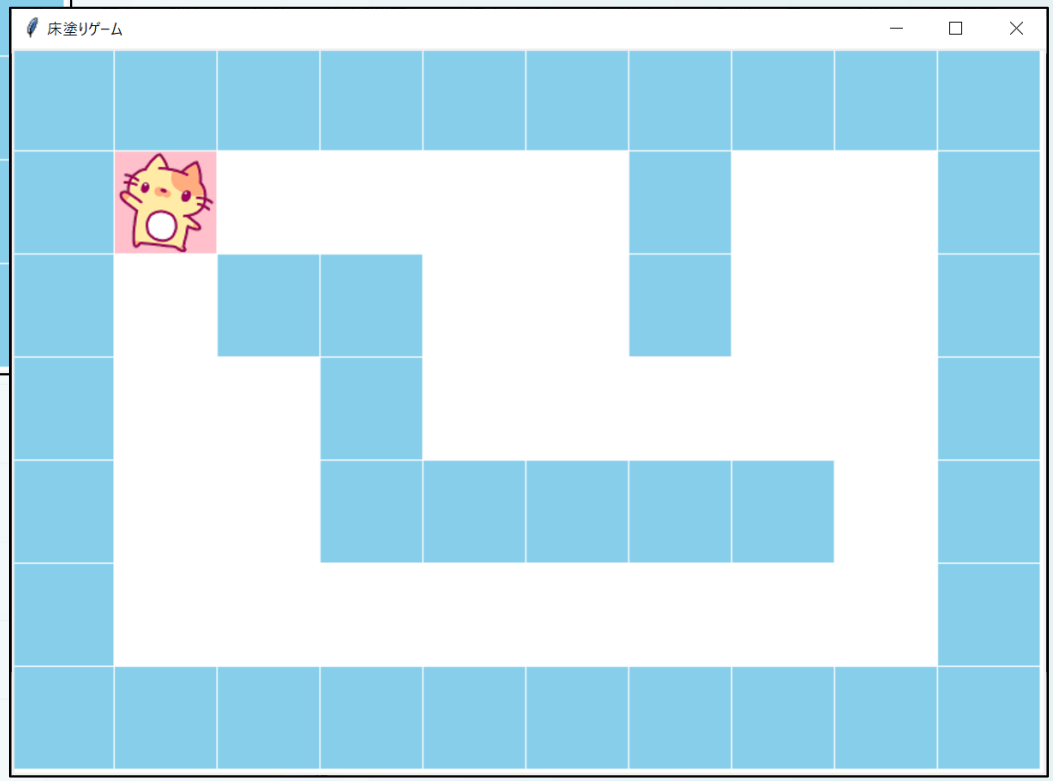
やり直せる処理を追加する

「Run Module」またはF5キーを押してプログラムを実行する。



左 Shift キーを押す

やり直せる！



復習＆チャレンジ

ここまで習ったことをScratchでもできるかチャレンジしてみよう。
その過程でScratchでできること、Pythonでないとできないことを整理してみよう。

メモ

プログラミング教室のテクノロ

なまえ：