プログラミング教室のテクノロ

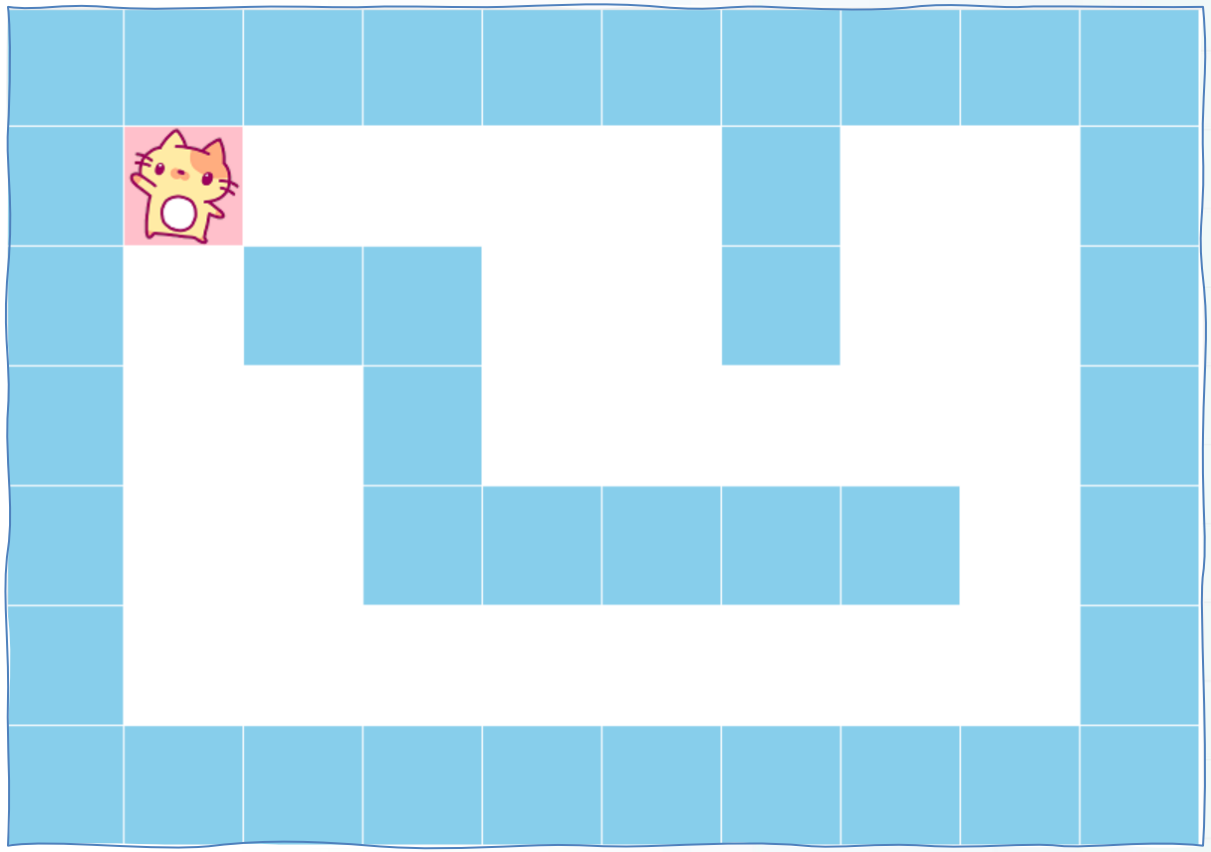
Pythonの道

本格的なゲーム開発①(前半)

もくじ

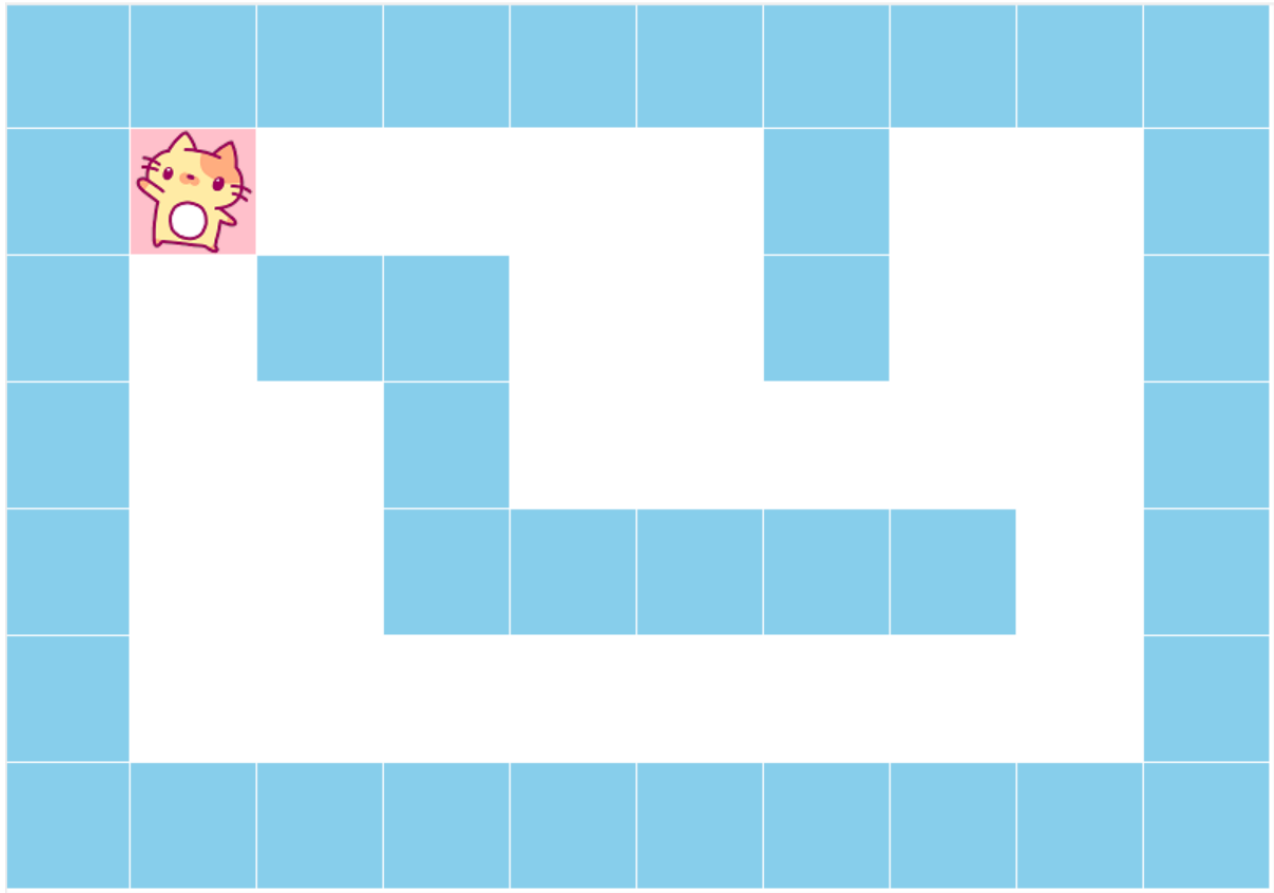
・床塗りゲームをつくる（前半）

　リアルタイムにキャラクタを動かす方法を学ぶ



本格的なゲーム開発の技術

ゲームソフトは常にキー入力を受け付け、画面を更新し続ける仕組みで動いている。これをリアルタイム処理という。キャラクターを動かして迷路の床を塗るゲームを作りながら本格的なゲーム開発に必要な技術を学ぶ。



after()命令を理解する

after()命令を使ってタイマーを作ってみよう。

```
import tkinter ・・・・tkinterモジュールの呼出し
tmr = 0 ・・・・変数「tmr」に0を代入する
def count_up(): ・・・count_up関数を呼び出した時に実行する関数を定義
    global tmr ・・・・「tmr」をグローバル変数として扱うと宣言
    tmr = tmr + 1 ・・・・「tmr」の値を1増やす
    label["text"] = tmr ・・・・ラベルに「tmr」の値を表示
    root.after(1000,count_up) ・・1秒(1000ミリ秒)後に再びこの関数を実行する
root = tkinter.Tk() ・・・ウィンドウのオブジェクトを作る
label = tkinter.Label(font=("system",80) ,text="スタート")
                                                ・・・ラベルの部品を作る
label.pack() ・・・ラベルの部品を配置する
root.after(1000,count_up) ・・1秒後に関数「count_up」を実行する
root.mainloop() ・・・ ウィンドウを表示する
```

after()命令を理解する

「Run Module」またはF5キーを押してプログラムを実行する。

実行するとスタートと表示されて１から順にカウントアップされていく

※「おみくじゲーム」や「ネコ度診断ゲーム」はボタンを押すことで初めて結果が表示され、ボタンを押さない限り画面に変化はない。ユーザが何かをして初めて処理が行われるものをイベントドリブン型と呼ぶことを知っておこう。

after()命令を理解する

after()命令はtkinterモジュールの中で使用できる命令で、処理を遅らせたり、定期的に実行したりするときに使う。

・after()命令の書き方

after(ミリ秒,実行する関数)

例

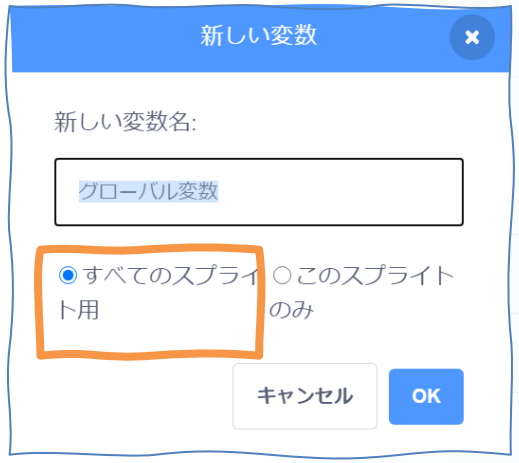
after(1000,テスト)

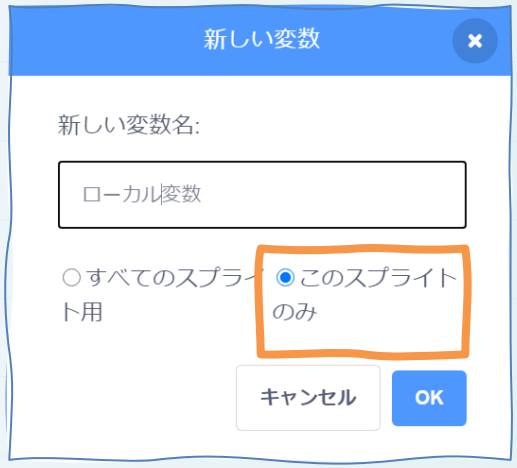
⇒1000ミリ秒後にテストという関数を実行する

※tkinterモジュールの中で使用するため、単独では使用できない。

グローバル変数を理解する

グローバル変数とは、複数の関数から使用できる変数のことを言う。 これに対して、ひとつの関数内でのみ使用できる変数のことをローカル変数という。





ネコもイヌも
「にゃー」という

ネコが「にゃー」という

イヌが「ワン」という

グローバル変数を理解する

```
ネコ1 = "にゃー"

def ローカル変数():
    ネコ2 = "にゃー"
    イヌ1 = "ワン"
    print(ネコ2)
    print(イヌ1)

ローカル変数()
print(ネコ1)
```

グローバル変数はどこでも使用できる

ローカル変数は関数の中でのみ使用できる

```
にゃー
ワン
にゃー
>>>
```

ローカル変数が呼び出される

グローバル変数が呼び出される

```
ネコ1 = "にゃー"

def ローカル変数():
    ネコ2 = "にゃー"
    イヌ1 = "ワン"
    print(ネコ2)
    print(イヌ1)

ローカル変数()
print(ネコ1)
print(ネコ2)
```

ローカル変数を関数の中以外で使うとエラーがでる

```
NameError: name 'ネコ2' is not defined
```

グローバル宣言すると関数の中以外でも使用できる

グローバル宣言

```
global ネコ2
```

グローバル変数とローカル変数

正しいコード

```
tmr = 0
def count_up():
    global tmr
    tmr = tmr + 1
    label["text"] = tmr
    root.after(1000,count_up)
```

コードを書き換えて確認しよう。
グローバル宣言が必要な理由を理解しよう。

エラーコード

```
tmr = 0
def count_up():
    global tmr
    tmr = tmr + 1
    label["text"] = tmr
    root.after(1000,count_up)
```

tmr = tmr + 1
でエラーが発生

エラーコード

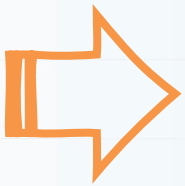
```
def count_up():
    tmr = 0
    tmr = tmr + 1
    label["text"] = tmr
    root.after(1000,count_up)
```

１のまま変わらない

キー入力を受け付ける

ゲームソフトではどのキーが押されているかを判定し、押されたキーに応じてキャラクタを動かしている。コンピューにキーがおされたことを認識させるプログラムを作ってみよう。

scratchで「〇〇キーが押されたとき」ブロックをPythonで作成していくイメージ



ユーザがシステムにキーやマウスを操作することをイベントという。

bind()命令を理解する

イベントを受け取るにはbind()命令を使う。キーイベントを取得するプログラムを作ってみよう。

・bind()命令の書き方

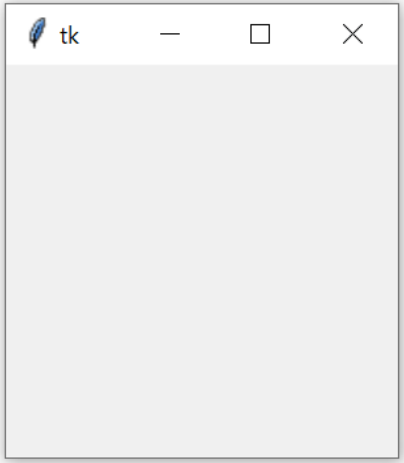
bind(“ <イベント> ” , イベント発生時に実行する関数名)

scratchの〇キーが押されたときブロックと同じ

<イベント>	イベントの内容
<KeyPress>	キーを押した
<Key>	
<KeyRelease>	キーを離した
<Motion>	マウスポインタを動かした
<ButtonPress>	マウスボタンをクリックした
<Button>	

```
import tkinter
def key_down(e):
    key = e.keycode
    print("キーが押されました")
root = tkinter.Tk()
root.bind("<KeyPress>",key_down)
root.mainloop()
```

```
キーが押されました
キーが押されました
キーが押されました
キーが押されました
キーが押されました
```



bind()命令を理解する

押されたキーのキーコードを表示するプログラムを作る。

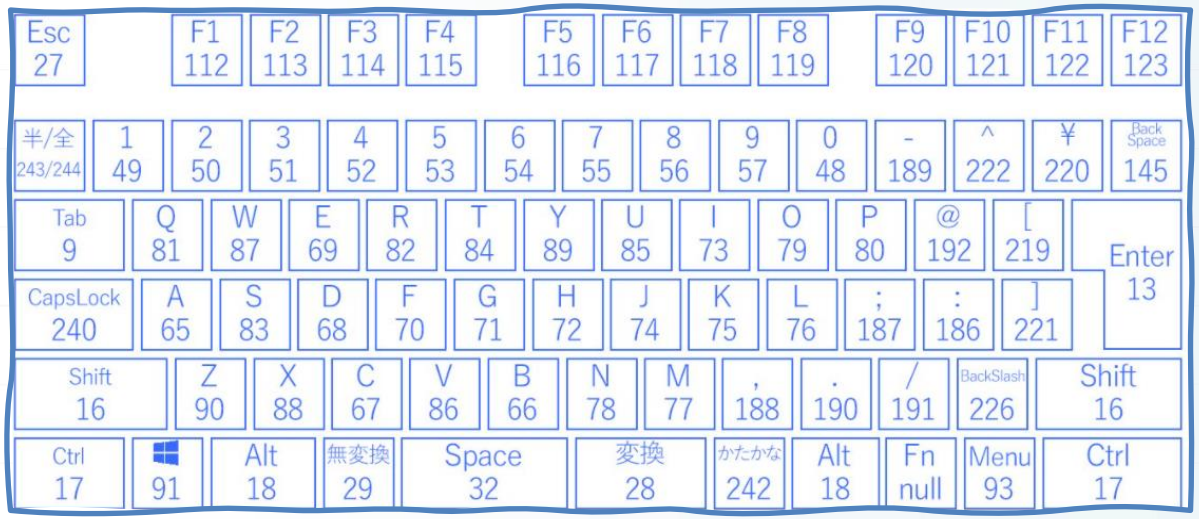
```
import tkinter  ・・・・tkinterモジュールの呼出し
def key_down(e): ・・・ key_down関数を呼び出した時に実行する関数を定義
    key = e.keycode ・・・ 押されたキーのコードを変数「key」に代入
    print("押されたボタンのキーコードは",key,"です。")
                                        ・・・ キーコードの表示
root = tkinter.Tk()  ・・・ ウィンドウのオブジェクトを作成
root.title("キーコードを取得") ・・・ ウィンドウのタイトルを表示する
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.mainloop()  ・ ・ ・ ウィンドウを表示する
```

bind()命令を理解する

「Run Module」またはF5キーを押してプログラムを実行する。

```
押されたボタンのキーコードは 49 です。
押されたボタンのキーコードは 50 です。
押されたボタンのキーコードは 51 です。
押されたボタンのキーコードは 52 です。
押されたボタンのキーコードは 53 です。
押されたボタンのキーコードは 54 です。
押されたボタンのキーコードは 55 です。
押されたボタンのキーコードは 56 です。
押されたボタンのキーコードは 57 です。
押されたボタンのキーコードは 48 です。
```

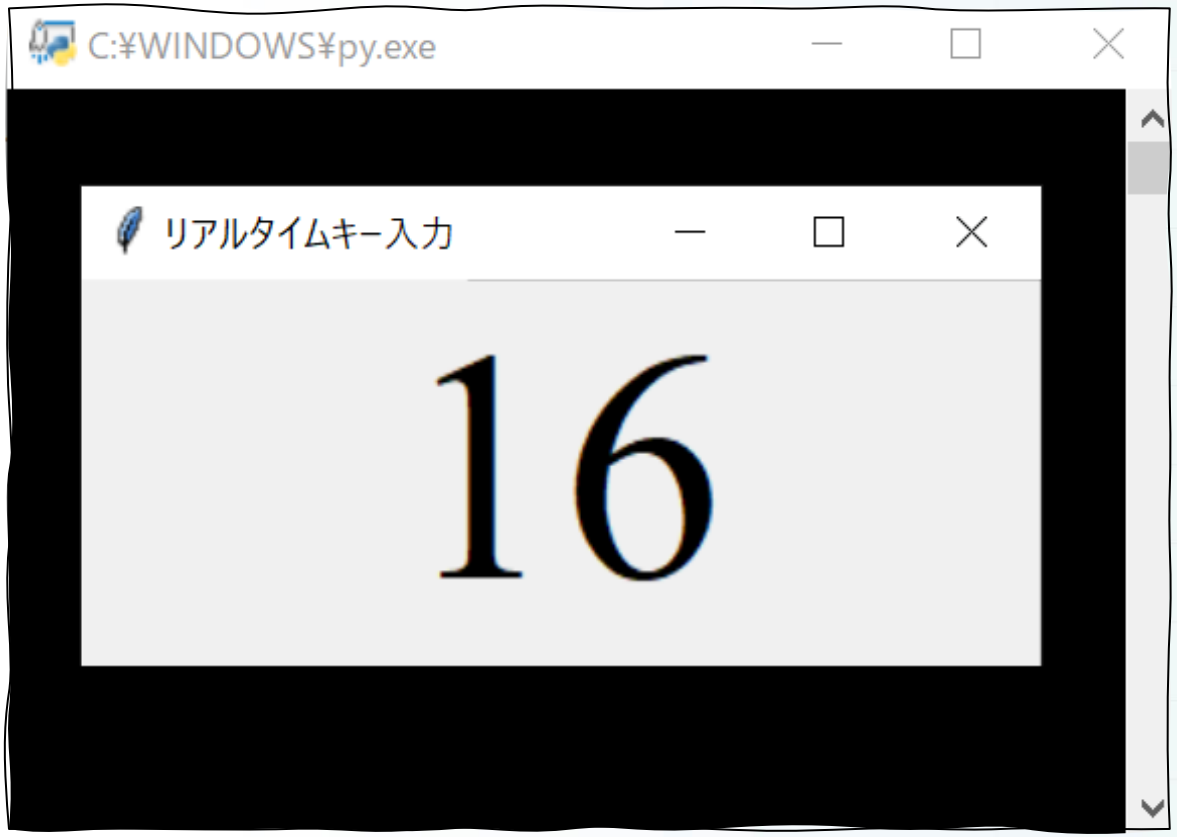
押されたキーのキーコードがシェルウィンドウに表示される。



キー入力で画像を動かす

タイマー作りで学んだリアルタイム処理とキーイベントの取得を同時に行うと画面に表示したキャラクターをキー入力で動かすことができるようになる。まずは、ウィンドウのラベル部分にキーコードが表示されるプログラムを作る。

押されたキーのキーコードがウィンドウのラベル部分に表示される。



リアルタイムキー入力

押されたキーのキーコードを表示するプログラム

```
import tkinter   ・・・・tkinterモジュールの呼出し
key = 0          ・・・・キーコードを入れる変数の宣言
def key_down(e):  ・・・・キーを押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 = e.keycode  ・・・・押されたキーのコードをKeyに代入
def main_proc():   ・・・・リアルタイム処理を行う関数を定義
    label["text"] = key  ・・・・ラベルにkeyの値を表示
    root.after(100, main_proc)・・・・after()命令で0.1秒後に実行する関数を指定
root = tkinter.Tk()  ・・・・ウィンドウのオブジェクトを作る
root.title("リアルタイムキー入力")・・・・ウィンドウのタイトルを指定
root.minsize(300,100)  ・・・・ウィンドウの初期サイズを指定
```

次のページに続く

リアルタイムキー入力

```
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

```
label = tkinter.Label(font=("system",100) ,text="スタート")
```

・・ラベルの部品(システムフォント、フォントサイズ100、テキスト欄にスタートと表示)を作る

```
label.pack()
```

・・・・ラベルの部品を配置する

```
root.after(1000,main_proc)
```

・・・after()命令で0.1秒後に実行する関数を指定

```
root.mainloop()
```

・・・・ウィンドウを表示する

「Run Module」またはF5キーを押してプログラムを実行する。

リアルタイムキー入力

キーコードではなく押されたキーの名称を表示させてみよう。
押されたキーの名称のことをキーシム(keysym)という。

```
import tkinter   ・・・・tkinterモジュールの呼出し
key = " "        ・・・・キーコードを入れる変数の宣言
def key_down(e):  ・・・・キーを押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 = e.keysym   ・・・・押されたキーのコードをKeyに代入
def main_proc():   ・・・・リアルタイム処理を行う関数を定義
    label["text"] = key  ・・・・ラベルにkeyの値を表示
    root.after(100, main_proc) ・・・・after()命令で0.1秒後に実行する関数を指定
root = tkinter.Tk()  ・・・・ウィンドウのオブジェクトを作る
root.title("リアルタイムキー入力")・・・・ウィンドウのタイトルを指定
root.minsize(300,100)  ・・・・ウィンドウの初期サイズを指定
```

次のページに続く

リアルタイムキー入力

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

label = tkinter.Label(font=(" system" ,50) ,text=" キー名称を表示")

・・ラベルの部品(システムフォント、フォントサイズ100、テキスト欄にキー名称を表示)を作る

label.pack() ・・・・ラベルの部品を配置する

root.after(1000,main_proc) ・・・after()命令で0.1秒後に実行する関数を指定

root.mainloop() ・・・・ウィンドウを表示する

「Run Module」またはF5キーを押してプログラムを実行する。

キー操作でキャラクターを動かす

ウィンドウ上に表示したキャラクターを、方向キーで上下左右に動かしてみよう。

```
import tkinter  ・・・・tkinterモジュールの呼出し
key = " "       ・・・・キーコードを入れる変数の宣言
def key_down(e):    ・・・・キーを押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 = e.keysym  ・・・・押されたキーのコードを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 = " "   ・・・・キーコードを入れる変数の宣言
cx = 400        ・・・・キャラクタのX座標を管理する変数
cy = 300        ・・・・キャラクタのY座標を管理する変数
```


次のページに続く

キー操作でキャラクターを動かす

```
def main_proc():
    global cx, cy
    if key == "Up":
        cy = cy - 20
    if key == "Down":
        cy = cy + 20
    if key == "Left":
        cx = cx - 20
    if key == "Right":
        cx = cx + 20
    canvas.coords("MYCHR", cx, cy)
    root.after(100, main_proc)
```

・・・・リアルタイム処理を行う関数を定義

・・・・cx,cyをグローバル変数として扱うと宣言

・・・・方向キーの上が押されたら

・・・・ｙ座標を20ドット減らす

・・・・方向キーの下が押されたら

・・・・ｙ座標を20ドット増やす

・・・・方向キーの左が押されたら

・・・・x座標を20ドット減らす

・・・・方向キーの右が押されたら

・・・・x座標を20ドット増やす

・・・・キャラクタを新しい座標に移動する

・・・after()命令で0.1秒後に実行する関数を指定

次のページに続く

キー操作でキャラクターを動かす

```
root = tkinter.Tk()
```
・・・・ウィンドウのオブジェクトを作る

```
root.title("キャラクターの移動")
```
・・・・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()命令でキーを押した時に実行する関数を指定する

```
canvas = tkinter.Canvas(width=800, height=600,
bg="lightgreen")
```
・・・ キャンバスの部品（横:800,縦:600、背景色:ライトグリーン）を作る

```
canvas.pack()
```
・・・・キャンバスを配置する

```
img = tkinter.PhotoImage(file="mimi.png")
```
・・・・キャラクタ画像を変数imgに代入

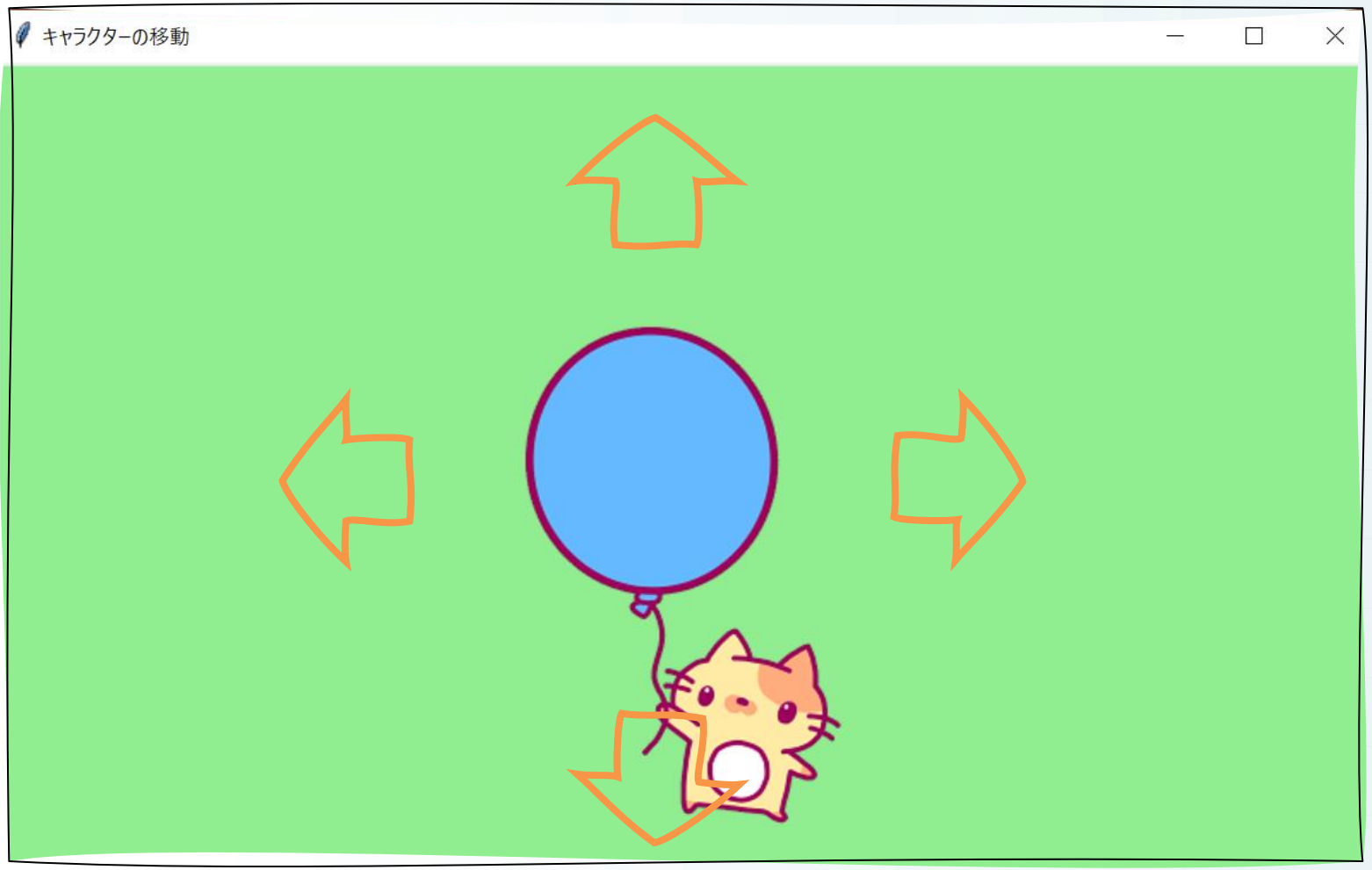
```
canvas.create_image(cx, cy, image=img, tag="MYCHR")
```
・・・・キャンバスに画像を表示

```
main_proc()
```
・・・・キャンバスに画像を表示

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

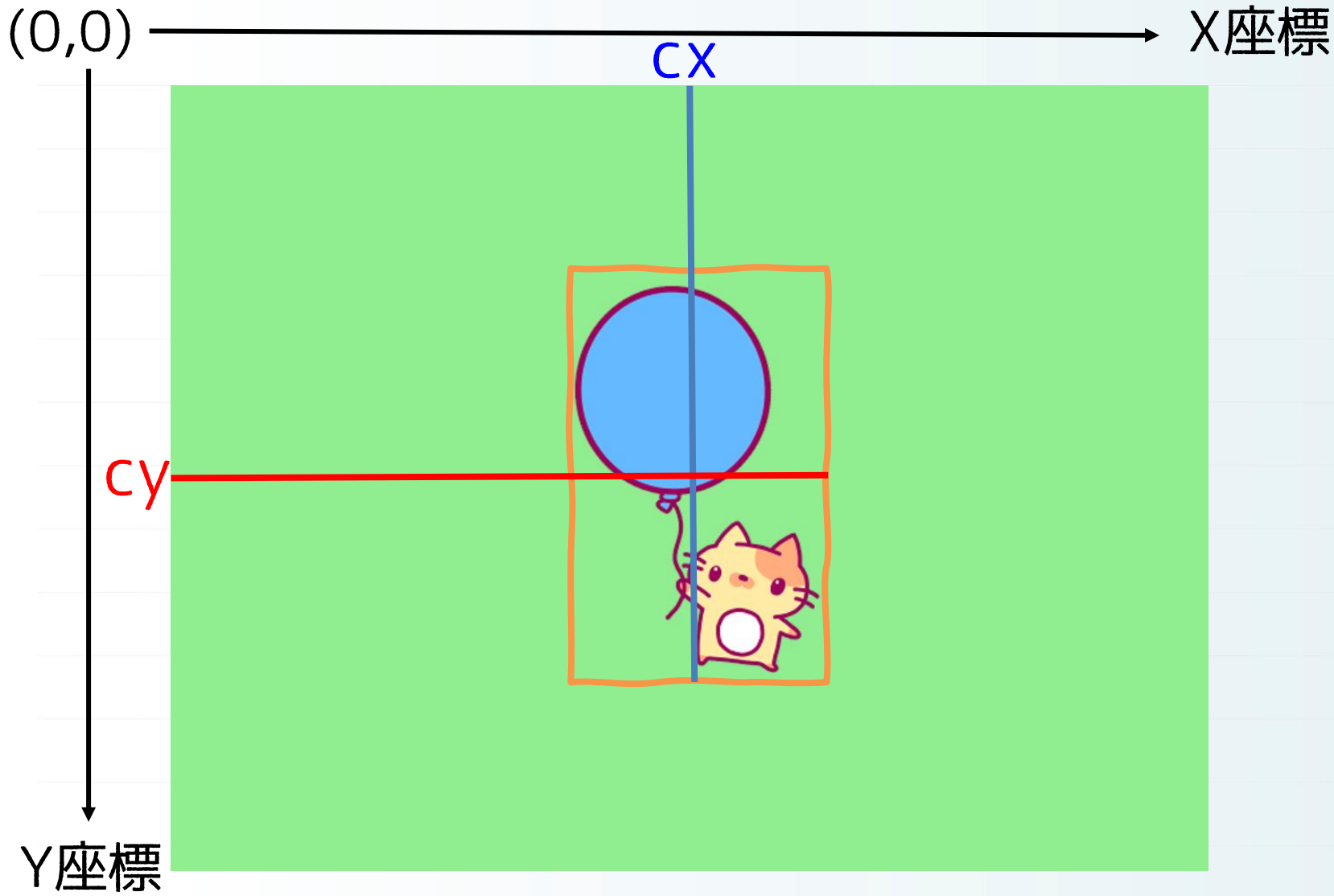
キー操作でキャラクターを動かす

「Run Module」またはF5キーを押してプログラムを実行する。



create_image()命令のおさらい

create_image()命令の引数の座標、つまりキャラクタの座標はcxとcyが交わる部分になる。



create_image()命令のおさらい

画像の表示はPhotoImage()命令で作り、create_image()命令で描画する。

・PhotoImage()命令の書き方

画像の変数名　＝ tkinter.PhotoImage(画像のファイル名)

画像のファイル名⇒(file ＝“ ファイル名.png”)

【例】

img ＝ tkinter.PhotoImage(file ＝“ XXX.png”)

・create_image()命令の書き方

⇒ create_image()命令で画像を表示させることができる

キャンバスの変数名.create_image(横幅 , 高さ , image ＝ 画像の変数名)

【例】

canvas.create_image(400 , 300 , image ＝ 画像)

タグについて理解しよう

create_image()命令の引数にタグをつけることができる。タグとはcreate_image()命令で作った図形や画像につける名前のこと。
お店の商品についているラベルのことを商品タグというようにcreate_image()命令で作った画像にもタグをつけて他の命令で呼び出すことができる。

・タグの書き方

```
canvas.create_image(cx , cy image = img , tag = " MYCHR" )
```

tag = の後に書いた" MYCHR" がタグ名になる。

タグをつけた図形や画像は他の命令を使って消したり、移動したりすることができる。

coords()命令を理解する

coordsとは座標を意味する言葉でcoords()命令は表示中の画像を新しい位置に移動するときに使用する。coords()命令の引数はタグ名、X座標、Y座標となる。

・coords()の書き方

X座標　Y座標　画像ファイルを格納した変数　タグ名

```
canvas.create_image(cx , cy  image = img , tag = " MYCHR" )
```

⇨ 最初にcanvas.create_image()命令でタグ付きの画像を作る

```
canvas.coords(" MYCHR"  , cx , cy)
```

移動させるタグ名　移動先のX座標　移動先のY座標

⇨ 移動させたいタグ名と移動先の座標を指定する。

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：