Python初心者必見！キーボード入力検知の10のステップ

⏳ この記事は**約21分**で読めます。

はじめに

Pythonでのプログラミングにおいて、キーボード入力を検知するというのは、あなたが作るアプリケーションがユーザーの入力を直接取得できるという意味で、重要なスキルとなります。

これによって、あなたのアプリケーションはユーザーのアクションに対応して動作を変更したり、特定のキー入力をトリガーとして機能を実行することが可能になります。

●Pythonとは

Pythonは、簡潔で読みやすいコードを書くことができる、人気の高いプログラミング言語の1つです。

その文法のシンプルさと汎用性の高さから、初心者から経験豊富なプロフェッショナルまで、幅広いユーザーに支持されています。

○Pythonの基本的な特性

Pythonは、そのコードが読みやすく、効率的なプログラム開発を可能にすることで知られています。

さらに、Pythonは豊富な標準ライブラリと、強力で多機能な外部ライブラリを備えています。

そのため、Pythonはさまざまなタスクに対応することができます。

●Pythonでのキーボード入力検知とは

Pythonでキーボード入力を検知するとは、Pythonプログラムがユーザーからのキーボードの入力を取得し、それに基づいて何かを実行することを意味します。

例えば、ユーザーが特定のキーを押すと、特定の関数が実行されるといったように、ユーザーの行動に応じてプログラムの動作を変更することができます。

○キーボード入力検知の仕組み

キーボード入力の検知は、ユーザーがキーボードを押すことで発生するイベントを捕捉することにより行われます。

Pythonでは、標準ライブラリの一部であるkeyboardを使用してキーボードイベントを検知することができます。

●Pythonでキーボード入力を検知するための手順

Pythonでキーボード入力を検知するための手順を紹介します。

○Python環境の準備

まず、Pythonがインストールされていることを確認し、必要ならインストールしてください。

Pythonの最新バージョンは公式ウェブサイトからダウンロードできます。

また、キーボード入力の検知にはkeyboardライブラリが必要なので、次のコマンドでインストールしてください。

```
pip install keyboard
```

コピー コードを実行

このコマンドを実行すると、Pythonのパッケージ管理システムであるpipがkeyboardライブラリをダウンロードし、Python環境にインストールします。

このライブラリを使用することで、Pythonプログラムがキーボード入力を検知できるようになります。

○サンプルコード1：キーボード入力の読み取り

キーボードからの入力を読み取る基本的なPythonコードを紹介します。

```
import keyboard

print("何かキーを押してください...")

while True:
    if keyboard.read_key():
        print("キーが押されました: " + keyboard.read_key())
        break
```

コピー コードを実行

このコードでは、keyboardライブラリを使って、ユーザーがキーボードの任意のキーを押すまで待機します。

キーが押されると、keyboard.read_key()関数がキーの名前を文字列として返し、そのキーの名前が表示されます。

その後、無限ループが終了し、プログラムも終了します。

このコードを実行すると、「何かキーを押してください…」と表示され、ユーザーがキーを押すと、押されたキーの名前が表示されます。

例えば、ユーザーが「a」キーを押すと、「キーが押されました: a」と表示されます。

○サンプルコード2：特定のキーの入力を検知する

Pythonでは、keyboard.is_pressed()関数を使用して特定のキーが押されているかどうかをチェックすることができます。

この関数にキーの名前を文字列として渡すと、そのキーが押されている場合はTrueを、押されていない場合はFalseを返します。

'q'キーが押されたときにプログラムを終了するコードを紹介します。

```
import keyboard

print("qキーを押すとプログラムが終了します...")

while True:
    if keyboard.is_pressed('q'):  # 'q'キーが押されているかチェック
        print("'q'キーが押されました。プログラムを終了します。")
        break
```

このコードでは、keyboard.is_pressed()関数を使って'q'キーが押されているかどうかをチェックしています。

コピー コードを実行

'q'キーが押されると、"qキーが押されました。プログラムを終了します。"と表示されてループが終了し、プログラムも終了します。

このコードを実行すると、"qキーを押すとプログラムが終了します…"と表示され、ユーザーが'q'キーを押すと、"'q'キーが押されました。プログラムを終了します。"と表示されてプログラムが終了します。

また、ユーザーが他のキーを押しても反応しないことがわかります。

これは、keyboard.is_pressed()関数がチェックしているのが'q'キーだけであり、他のキーが押されても無視されるためです。

さらに発展させて、ユーザーが押したキーに応じて異なる動作をするプログラムも作成できます。

例えば、'a'キーが押されたら"Hello"と表示し、'b'キーが押されたら"World"と表示するプログラムは次のようになります。

```
import keyboard

print("'a'キーを押すと'Hello'を表示します。'b'キーを押すと'World'を表示します。")

while True:
    if keyboard.is_pressed('a'):  # 'a'キーが押されているかチェック
        print("Hello")
    elif keyboard.is_pressed('b'):  # 'b'キーが押されているかチェック
        print("World")
    elif keyboard.is_pressed('q'):  # 'q'キーが押されているか

チェック
        print("'q'キーが押されました。プログラムを終了します。")
        break
```

このコードを実行すると、ユーザーが'a'キーを押すと"Hello"が、'b'キーを押すと"World"が表示されます。また、'q'キーを押すとプログラムが終了します。

コピー コードを実行

これにより、キーボードの特定のキーによるプログラムの制御が可能となります。

○サンプルコード3：連続的なキーボード入力を検知する

前回のステップでは、特定のキーの押下を検知する方法を紹介しました。

今回はそれを発展させ、ユーザーがキーボードを連続的に打鍵している状況を検知する方法について解説します。

キーボードの任意のキーが押され続けている状況を検知するサンプルコードを紹介します。

```
import keyboard
import time

print("任意のキーを押し続けると検知します。")

start_time = time.time()
while True:
    keys = keyboard.get_pressed_keys()  # 現在押下されているキーを取得
    if keys:  # 何らかのキーが押されている場合
        elapsed_time = time.time() - start_time
        print(f"{elapsed_time}秒間、キー{keys}が押され続けています。")
        start_time = time.time()
```

コピー コードを実行

このコードでは、keyboard.get_pressed_keys()関数を使用して、現在押下されているキーのリストを取得しています。

その後、何らかのキーが押下されている（つまり、取得したキーのリストが空でない）場合、そのキーが押下されてからの経過時間を計算し、結果を表示しています。

実行すると、ユーザーがキーボードの任意のキーを押し続けると、「〇〇秒間、キー['a']が押され続けています。」のようなメッセージが表示されます。

ここで、「a」はユーザーが押し続けたキーを表しています。

このように、keyboard.get_pressed_keys()関数を使用することで、キーボードの連続的な入力を検知することが可能です。

〇サンプルコード4：非同期にキーボード入力を検知する

非同期処理を利用することで、キーボード入力の検知をバックグラウンドで行いつつ、他の処理を前面で行うことが可能になります。

```python
import keyboard
import time

def key_detected(e):
    print(f'キー{e.name}が押されました')

keyboard.on_press(callback=key_detected)

while True:
    time.sleep(1)
    print('この処理はキーボード入力の検知と並行して実行されます')
```

コピー コードを実行

このコードでは、keyboard.on_press(callback=key_detected)という構文を使って、キーボードが押されるたびにkey_detected関数が呼び出されるように設定しています。

key_detected関数はキーの情報を引数に取り、押されたキーの名前を出力します。

その後、無限ループ内で一定時間ごとにメッセージを出力する処理が行われています。

このメッセージの出力はキーボード入力の検知と並行して実行されます。

実行すると、「この処理はキーボード入力の検知と並行して実行されます」というメッセージが一秒ごとに出力されます。

その間にキーボードの任意のキーを押すと、「キー〇〇が押されました」というメッセージが表示されます。

これにより、非同期にキーボード入力の検知が行われていることが確認できます。

次に、Pythonでキーボード入力を検知する具体的な応用例をいくつか紹介します。

これらの例は、Pythonのキーボード入力検知機能を使って実現できる幅広い用途を示しています。

●Pythonでキーボード入力を検知する応用例

キーボード入力の検知は、ゲームの操作、特定のキーをトリガーとした処理の実行、キーボードマクロの作成、キーボードログの作成、ホットキーの設定、特定のキーの入力をブロックするなど、様々な場面で利用できます。

これらの応用例について順に詳しく説明していきます。

○サンプルコード5：ゲームの操作

まず、ゲームの操作について考えてみましょう。

キーボードの入力を検知することで、ユーザーがキーボードを使ってゲームキャラクターを操作することが可能になります。

たとえば、Wキーで前進、Aキーで左旋回、Dキーで右旋回、Sキーで後退といった操作が可能です。

それでは、具体的なコードを見てみましょう。

コピー コードを実行

```
import keyboard

class Player:
    def __init__(self):
        self.x = 0
        self.y = 0

    def move_up(self):
        self.y += 1
        print(f'プレイヤーの位置: ({self.x}, {self.y})')

    def move_down(self):
        self.y -= 1
        print(f'プレイヤーの位置: ({self.x}, {self.y})')

    def move_left(self):
        self.x -= 1
        print(f'プレイヤーの位置: ({self.x}, {self.y})')

    def move_right(self):
        self.x += 1
        print(f'プレイヤーの位置: ({self.x}, {self.y})')

player = Player()

keyboard.on_press_key("w", lambda _: player.move_up())
keyboard.on_press_key("s", lambda _: player.move_down())
keyboard.on_press_key("a", lambda _: player.move_left())
keyboard.on_press_key("d", lambda _: player.move_right())

keyboard.wait()
```

このサンプルコードでは、簡易的なゲームのプレイヤーを操作するコードを紹介しています。

プレイヤーはPlayerクラスのオブジェクトとして表現され、xとyの二つの座標を持っています。

move_up、move_down、move_left、move_rightの4つのメソッドにより、プレイヤーの位置を移動させることができます。

このコードでは、Wキーを押すとプレイヤーが上に、Aキーを押すと左に、Sキーを押すと下に、Dキーを押すと右に移動します。

実際に上記のコードを実行すると、各キーの入力に応じてプレイヤーの位置が変わり、「プレイヤーの位置: (x, y)」と表示されます。

これにより、キーボードの入力によるゲームの操作を実現しています。

○サンプルコード6：特定のキーをトリガーとした処理の実行

特定のキーをトリガーとした処理の実行について説明します。

ここでは、キーボードの特定のキーが押されたときに、特定の処理を自動的に実行する例を挙げます。

このような機能は、例えば音楽ソフトウェアでキーボードを使った楽器の演奏など、さまざまな用途で利用できます。

コピー コードを実行

```
import keyboard

def greeting():
    print("Hello, Python初心者！")

keyboard.add_hotkey('space', greeting)

keyboard.wait()
```

上記のコードでは、スペースキーをトリガーとして特定の関数を呼び出す例を示しています。

具体的には、スペースキーが押された時にgreeting関数が実行されます。

greeting関数は、"Hello, Python初心者！"というメッセージを出力するシンプルな関数です。

このコードを実行すると、スペースキーを押すたびにコンソール上に"Hello, Python初心者！"と表示されます。

これにより、特定のキーをトリガーとして任意の処理を実行する方法を理解できます。

○サンプルコード7：キーボードマクロの作成

キーボードマクロとは、一連のキー操作を記録して、それを再生することで一連の操作を自動化する機能のことを指します。

この機能は効率的な作業を可能にします。

```
import keyboard

keyboard.start_recording()
# ここでキーボード操作を行います。
events = keyboard.stop_recording()
keyboard.play(events)
```

コピー コードを実行

このコードでは、キーボードマクロの作成と再生を行っています。

start_recording関数でキーボード操作の記録を開始し、その後に行われるキーボード操作を全て記録します。

記録を終えるにはstop_recording関数を使用します。この関数は記録したキーボード操作（イベント）のリストを返します。

最後に、play関数を用いて記録した操作を再生します。

コードを実行した後に任意のキーボード操作を行ってみてください。

その操作が終わったら記録を停止し、その操作が再現されるのを確認します。

○サンプルコード8：キーボードログの作成

この機能は、あなたが何を入力したかを追跡するのに役立ちます。

ただし、この技術は不適切に使用するとプライバシーの侵害につながる可能性があるため、倫理的な範囲内で使用することが非常に重要です。

コピー コードを実行

```
import keyboard

# 記録ファイルの名前
log_file = "key_log.txt"

# イベントが発生したらログを保存する関数
def log_event(e):
    with open(log_file, "a") as f:
        f.write(str(e) + "\n")

# グローバルにイベントを記録
keyboard.hook(log_event)

# 無限ループ
while True:
    pass
```

このコードではキーボードログの作成を行っています。

具体的には、全てのキーボードイベントを記録して、それらをテキストファイルに保存します。

ログファイルの名前は変数log_fileで指定します。

関数log_eventは、キーボードイベントが発生した時に呼び出され、そのイベントをログファイルに書き込みます。hook関数は、全てのキーボードイベントを捕捉するために使われます。

指定した関数（この場合はlog_event）がキーボードイベントのたびに呼び出されます。

このコードを実行すると、key_log.txtファイルが作成され、その中に全てのキーボードイベントが記録されます。

ただし、このコードを終了させるためには、ターミナルやコマンドプロンプトを閉じる、またはプログラムの実行を中断する必要があります。

○サンプルコード9：ホットキーの設定

ホットキーとは、特定のキーの組み合わせを押すことで特定の操作を実行するためのショートカットキーのことです。

例えば、多くのソフトウェアではCtrl+Cを押すことでコピー操作を行います。これはホットキーの一例です。

このスキルを使えば、自分だけのカスタムショートカットを作ることができます。

コピー コードを実行

```
import keyboard

# ホットキーの設定
hotkey = 'ctrl + shift + a'

# ホットキーが押された時の動作
def action(e):
    print('ホットキーが押されました！')

# ホットキーの設定
keyboard.add_hotkey(hotkey, action)

# 無限ループ
while True:
    pass
```

このコードでは、Pythonのkeyboardライブラリを用いて、ホットキー（ショートカットキー）の設定を行っています。

この例では、ctrlキーとshiftキーとaキーを同時に押すことで、actionという名前の関数が呼び出される設定をしています。

ホットキーの組み合わせは、hotkeyという変数で定義しています。

この組み合わせは自由に変更可能で、他のキーの組み合わせでも構いません。

例えば、ctrl + alt + delやshift + f1などに変更することも可能です。

action関数は、設定したホットキーが押された時に実行される関数です。

この関数は何でも良いので、ホットキーを押したときに行いたい処理を記述します。

この例では、ホットキーが押されました！というメッセージを出力します。

keyboard.add_hotkey関数は、ホットキーの組み合わせと、そのホットキーが押されたときに呼び出される関数を引数として受け取ります。

これにより、指定したホットキーが押されると、指定した関数が実行されるようになります。

このコードを実行すると、ctrl + shift + aを押すと、ホットキーが押されました！と表示されます。

ただし、このコードを終了するためには、ターミナルを閉じるか、プログラムの実行を中断する必要があります。

○サンプルコード10：特定のキーの入力をブロックする

特定のキーの入力をブロックする方法について解説します。

これは、特定のキーが押されたときに、そのキーの入力を無効にする機能です。

例えば、ゲームプレイ中に誤ってゲームを終了させてしまうescキーの入力を無効にするなど、様々な用途で活用することができます。

コピー　コードを実行

```
import keyboard

# ブロックするキーの設定
block_key = 'esc'

# キーの入力をブロックする関数
def block_key(e):
    return False

# キーの入力をブロックする設定
keyboard.on_press_key(block_key, block_key)

# 無限ループ
while True:
    pass
```

このコードでは、Pythonのkeyboardライブラリを使って、特定のキーの入力をブロックする設定を行っています。

この例では、escキーが押された時に、その入力を無効にする設定をしています。

ブロックしたいキーはblock_keyという変数で定義しています。この変数は自由に変更可能で、他のキーをブロックすることも可能です。

例えば、f1やaltなどのキーの入力をブロックすることも可能です。

block_key関数は、特定のキーが押されたときにその入力をブロックするための関数です。

この関数はFalseを返すことで、キーの入力を無効にします。

keyboard.on_press_key関数は、特定のキーが押されたときに、そのキーの入力をブロックする設定を行います。

引数として、ブロックしたいキーと、そのキーが押されたときに呼び出される関数を指定します。

このコードを実行すると、escキーを押しても何も反応しなくなります。

ただし、このコードを終了するためには、ターミナルを閉じるか、プログラムの実行を中断する必要があります。

このようにPythonでは、キーボード入力を検知したり、ホットキーを設定したり、特定のキーの入力をブロックしたりすることが可能です。

●キーボード入力検知の注意点と対処法

キーボード入力検知は便利な機能ですが、次のような注意点があります。

まず、プライバシーの観点から問題がある可能性があります。

たとえば、入力を監視することで、ユーザーの個人情報を取得するリスクがあるため、プログラムがユーザーの承諾を得て使用されることが重要です。

キーボード入力を監視するプログラムを作成する際は、プライバシー保護の観点から、ユーザーが知らない間に情報を取得しないよう、適切な使用方法と目的をユーザーに伝えることが必要です。

また、キーボード入力検知は他のプログラムやOSとの互換性の問題を引き起こす可能性があります。

特定のキーの入力をブロックしたり、システムレベルでキーボード入力を監視すると、他のプログラムがキーボード入力を正しく検知できなくなる可能性があります。

また、OSによっては、キーボード入力を監視するための特殊な権限が必要な場合があります。

これらの問題を避けるために、次のような対処法があります。

1. ユーザーに対して、プログラムが行う行為と目的を明確に伝え、同意を得る。
2. プログラムがシステム全体に影響を与えないように、特定のアプリケーション内でのみキーボード入力を監視する。
3. 必要な権限を取得してからキーボード入力を監視する。

以上のように、キーボード入力検知の機能は便利ですが、同時にそれなりの注意が必要です。

Pythonを使ってキーボード入力を扱う際には、これらの点を心に留めておきましょう。

●キーボード入力検知のカスタマイズ方法

Pythonのkeyboardライブラリは非常に柔軟性が高く、様々な方法でキーボード入力を検知し、それに対応するアクションを実行することが可能です。

特定のキーが押されたときに特定の関数を実行する、複数のキーの組み合わせを検知するなど、さまざまなカスタマイズが可能です。

ここでは、それらのカスタマイズ方法をいくつか紹介します。

まず、「A」キーが押されたときにメッセージを表示するプログラムを見てみましょう。

コピー コードを実行

```
import keyboard

# 'A'キーが押されたときに実行する関数を定義
def print_message(e):
    print('Aキーが押されました。')

# 'A'キーの押下を監視
keyboard.on_press_key('A', print_message)

# 無限ループでイベントを監視
keyboard.wait()
```

このコードではkeyboard.on_press_key関数を使って'A'キーが押されたときにprint_message関数を呼び出します。

そして、keyboard.wait()によって無限ループでイベントを監視しています。

また、複数のキーを同時に押下したときに特定のアクションを実行するというカスタマイズも可能です。

下記の例では、「Ctrl」と「C」キーを同時に押した場合にメッセージを表示します。

コピー コードを実行

```
import keyboard

# 'ctrl'と'C'キーが同時に押されたときに実行する関数を定義
def print_message(e):
    print('Ctrl+Cが押されました。')

# 'ctrl'と'C'キーの同時押下を監視
keyboard.add_hotkey('ctrl+c', print_message)

# 無限ループでイベントを監視
keyboard.wait()
```

このコードでは、keyboard.add_hotkey関数を使って'ctrl'キーと'C'キーの同時押下を監視しています。

この関数は第一引数にホットキー（特定のキーの組み合わせ）を指定し、第二引数にそのホットキーが押下されたときに実行する関数を指定します。

これらのカスタマイズ方法を利用すれば、より細かいキーボード入力の検知とそれに対応するアクションをプログラムに組み込むことができます。

まとめ

今回の記事では、Pythonを用いたキーボード入力の検知について詳細に解説しました。

keyboardライブラリのインストールから、基本的なキーボード入力の検知方法、さらには注意点やカスタマイズ方法についてまで、一通りの知識を得ることができたと思います。

Pythonはその柔軟性から多岐にわたる用途で利用されていますが、キーボード入力の検知という面では特にその能力を発揮します。

特定のキーの押下や複数のキーの同時押下を検知することで、ユーザーの操作により直感的に反応するプログラムを作ることが可能となります。

しかし、その一方でキーボード入力の検知はOSや実行環境による影響を受けやすいため、確実に機能するプログラムを作るためには注意が必要です。

特にkeyboardライブラリは管理者権限が必要な場合がありますので、その点は十分に注意しましょう。

今回紹介したサンプルコードやカスタマイズ方法は、Pythonでキーボード入力を検知するための基本的なものです。

これをベースに、より複雑なキーボード入力の検知やそれに応じたアクションを作り出すことで、Pythonの世界はさらに広がります。

Python初心者の方でも理解しやすいように説明しましたが、実際にコードを書き、実行することでより深い理解が得られます。

手を動かして実践的な学習を行い、Pythonのキーボード入力検知の世界を探索してみてください。

目次 >

目次 [非表示]