2022年7月10日現在

サポート情報	たった2日でマスターするiPhoneアプリ開発集中講座 Xcode 13 / iOS 15 / Swift 5.5 対応

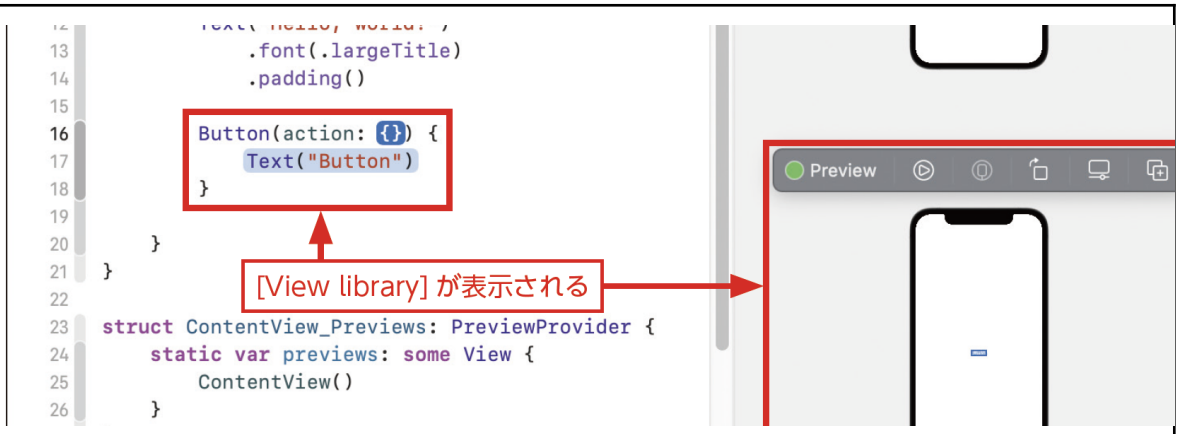
本書の出版時は、Xcode13.0 での動作確認を行っていますが、Xcode・Swift・iOS のバージョンアップに伴い、本書の記載内容やサンプルアプリに変更が必要なときがあります。
変更が必要な箇所を、このサポート情報にてご連絡いたします。

ご使用されている **Xcode** のバージョンをご確認の上で、このサポート情報を利用してください。
また、正誤も合わせて記載させて頂きます。

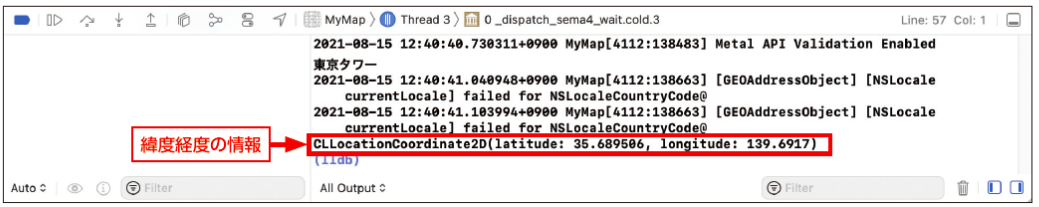
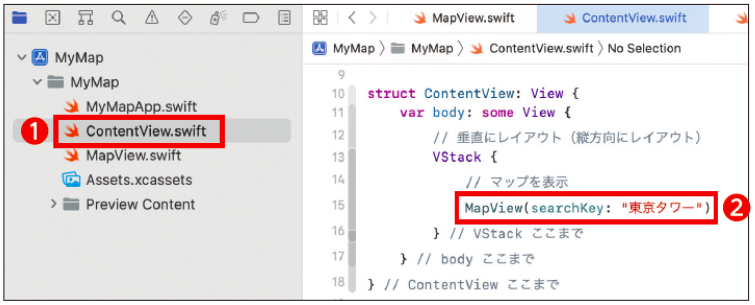
■変更内容：

執筆時点では問題がありませんでしたが、Xcodeのバージョンアップ やApple社の影響に伴い変更になった箇所です。ご自身の**Xcode**のバージョンを確認した上で、次の変更箇所を参考にしてください。

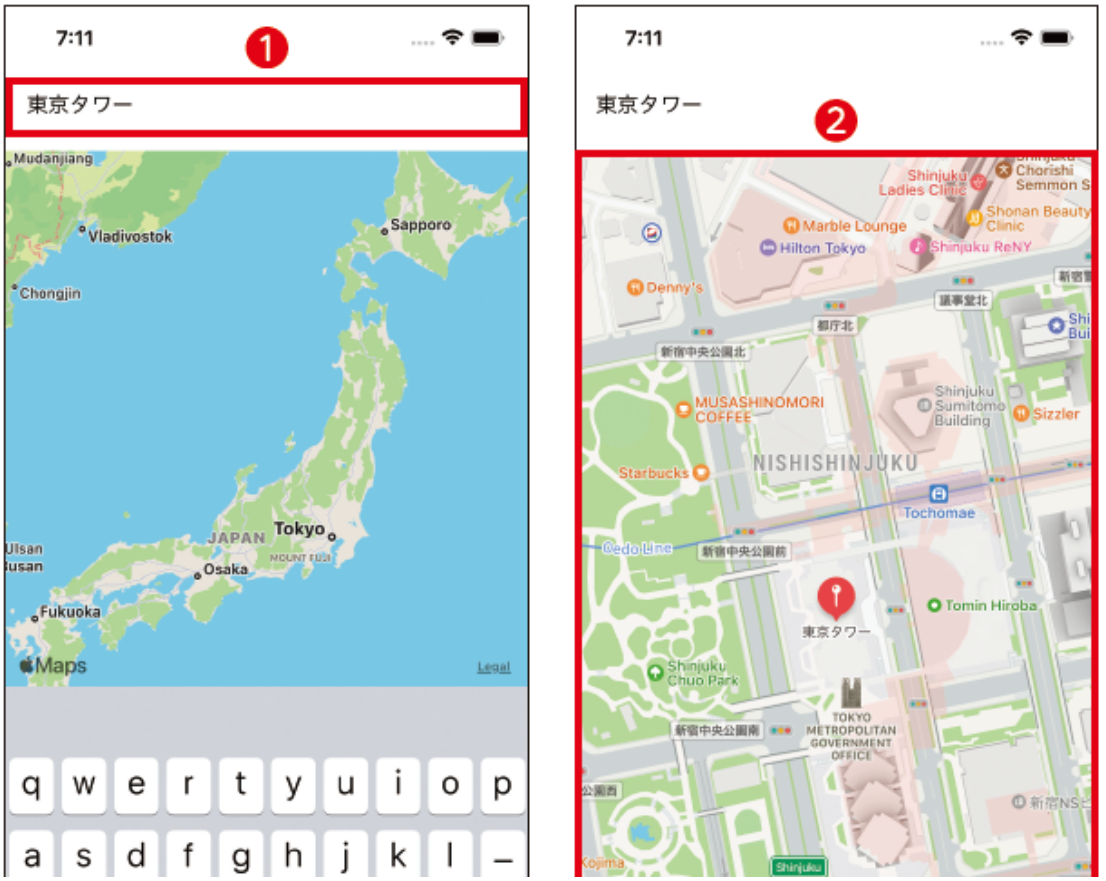
P.67：図「追加された Button」で挿入されたコードに変更があります。

Xcode13.0	
Xcode13.2.1	以下のように本書とは異なったコードが挿入されます。  上記の追加されたコードを、本書と同じコードになるように手で修正してから学習を継続してください。

P.209：図「デバッグエリアでの緯度経度情報の確認」で緯度経度が出力されない

Xcode全般	print文で指定した緯度経度の情報がデバッグエリアに出力されない。 ▼ デバッグエリアでの緯度経度情報の確認  シミュレータを起動して、ここまでで入力したプログラムの動きを確認します。 デバッグエリアの最後に、緯度経度情報が表示されています。「CLLocationCoordinate2D」に入っている「latitude」が緯度、「longitude」が経度です。
Xcode全般	**P207**で入力した「東京タワー」では、正常に緯度経度ができないときがあるようです。「羽田空港」であれば正常に取得できますので、ワードを変更して学習を継続してください。 「MyMapApp.swift」から起動された、ContentViewを確認してみましょう。 ❶ 「ContentView.swift」を選択してファイルを開きます。 ❷ ContentViewの中では、MapViewが実行さ ▼ ContentViewで、MapViewを呼び出し  執筆時点では、検索ワードが「東京タワー」で問題なく緯度経度が取得できていました。

P.227：図「キーワードで検索」で異なる位置にピンが立っている

Xcode全般	検索窓に「東京タワー」と入力しているのに、マップ上では異なる場所にピンが立っています。 ▼ キーワードで検索 
Xcode全般	本書の誤りではありますが、**Xcode**の不具合でもあります。 シミュレータ・プレビューでは正常にピンが立たないことがあります。検索ワードを「羽田空港」のように他のワードで試してください。

P.443：デバッグエリアでエラーが表示される

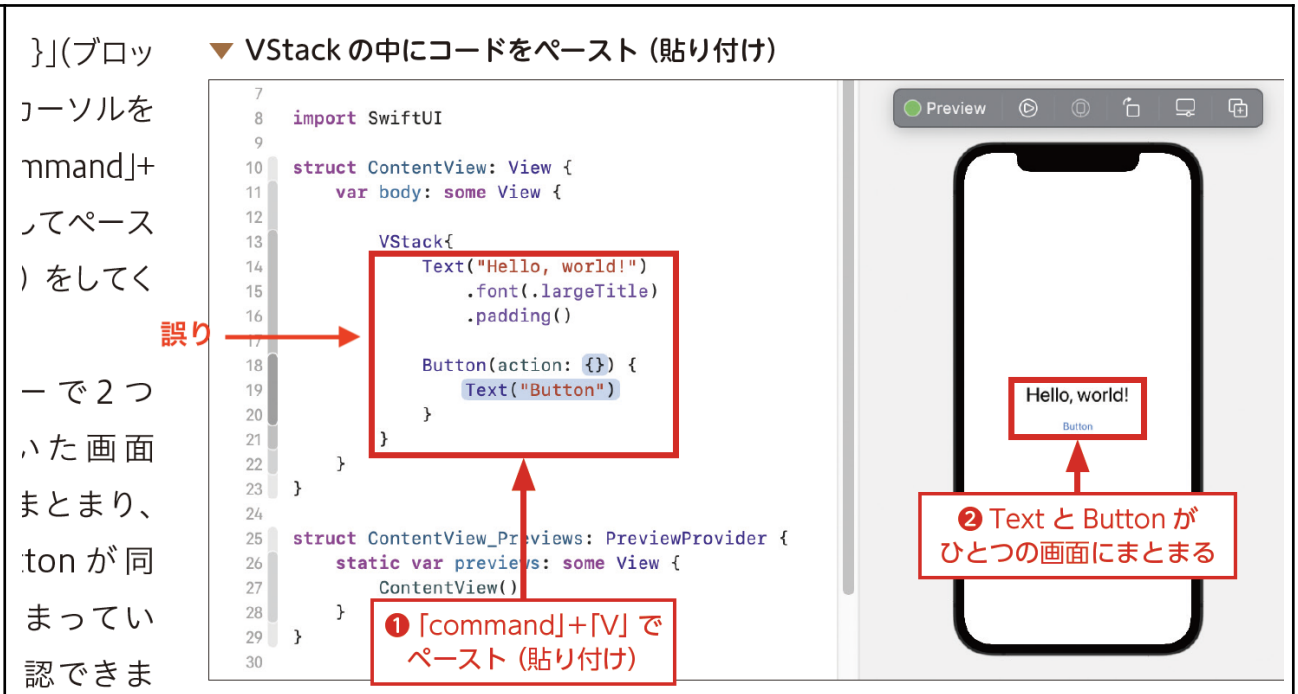
Xcode13以降	「2-6　再度シミュレータを起動して、デバッグエリアを確認」にて、デバッグエリアで次のエラーが表示される。 2022-03-27 16:37:04.684462+0900 MyOkashi[41495:3281982] [boringssl] boringssl_metrics_log_metric_block_invoke(153) Failed to log metrics
	Xcodeのバージョンアップに伴いエラーが表示されるようになりました。このエラーは無視しても大丈夫です。エラーが表示されていてもアプリは正常に動いています。

■正誤表：

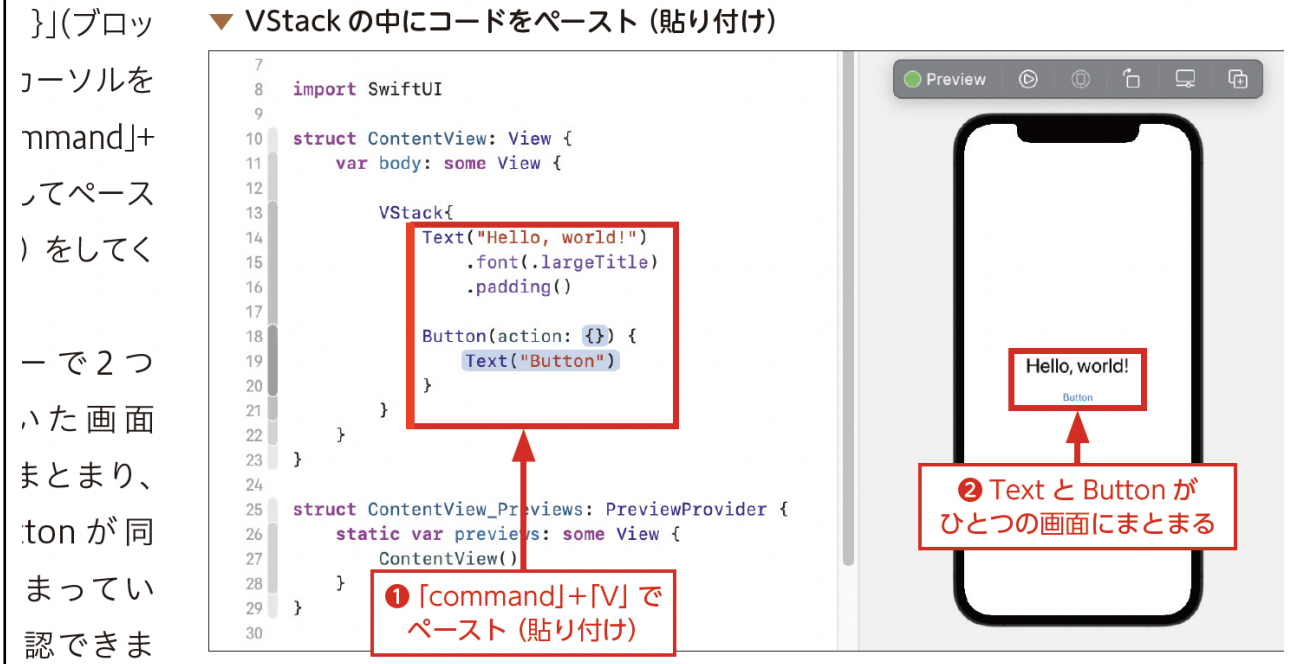
本書において下記のとおり、誤りがございました。
恐れ入りますが、本正誤表をご確認の上、ご利用いただきますようお願い申し上げます。

P.70：図「VStackの中にコードをペースト（貼り付け）」にて赤枠の範囲に誤りがあります。

【誤】



【正】



P.139:「コード解説」にて「関数」と記載していますが、「メソッド」が正しいです。

【誤】	「Int.random」は、Int 型で用意されている、random（ランダム）関数を使うことを示しています。 random は、数値をランダムに算出する関数（かんすう：**Function**）です。関数はある処理を行う、まとまったコードを切り出したものを指します。 今回のrandom 関数の中では、乱数を計算するためにたくさんの処理が動いていますが、私たちは、random を指定するだけで、乱数を生成させることができます。 random 関数に指定している「in: 1...3」は「引数」と呼ばれます。「in:」は、引数のラベルで、引数がどんな値なのかを表しています。 random 関数の使い方は、「random(in: 1...3)」のように、発生させたい数の範囲を引数に指定します。
【正】	「Int.random」は、Int 型で用意されている、random（ランダム）メソッドを使うことを示しています。 random は、数値をランダムに算出するメソッド（**Method**）です。~~関数はある処理を行う、まとまったコードを切り出したものを指します。~~ 今回のrandom メソッドの中では、乱数を計算するためにたくさんの処理が動いていますが、私たちは、random を指定するだけで、乱数を生成させることができます。 random メソッドに指定している「in: 1...3」は「引数」と呼ばれます。「in:」は、引数のラベルで、引数がどんな値なのかを表しています。 random メソッドの使い方は、「random(in: 1...3)」のように、発生させたい数の範囲を引数に指定します。

P.139:「Point」にて「関数」と記載していますが、「メソッド」が正しいです。

【誤】	今回は、Int 型のrandam 関数を利用しましたが、同じ名前の関数でも様々な種類があります。 Int 型のrandam でも、引数が1 つの関数や、2 つ関数もあります。 また、Bool 型をランダムに発生させる、Bool.random() のように、引数のない関数もあります。 引数の有無、種類によって違う動きの関数になります。大切なことなので覚えておいてください。
【正】	今回は、Int 型のrandam メソッドを利用しましたが、同じ名前のメソッドでも様々な種類があります。 Int 型のrandam でも、引数が1 つのメソッドや、2 つメソッドもあります。 また、Bool 型をランダムに発生させる、Bool.random() のように、引数のないメソッドもあります。 引数の有無、種類によって違う動きのメソッドになります。大切なことなので覚えておいてください。

P.152:「COLUMN 命名規則」の解説に誤りがあります。

【誤】	今回の変数のように、最初の文字を小文字から初めて、単語の区切りの文字を大文字にする規則を、lowerCamelCase（ローワーキャメルケース）といいます。 逆に最初の文字を大文字から初めて、以降の単語の区切りも大文字にする規則を、UpperCamelCase（アッパーキャメルケース）といいます。
【正】	今回の変数のように、最初の文字を小文字から始めて、単語の区切りの文字を大文字にする規則を、lowerCamelCase（ローワーキャメルケース）といいます。 逆に最初の文字を大文字から始めて、以降の単語の区切りも大文字にする規則を、UpperCamelCase（アッパーキャメルケース）といいます。

P.160:「COLUMN プログラミングスコープ」の解説に誤りがあります。

【誤】	［上から四行目］ 上記は、変数count が10 より大きければprint 文でメッセージを出力しています。
【正】	［上から四行目］ 上記は、定数count が10 より大きければ、定数**display**の内容をprint文で出力しています。

P.190:「COLMUN オプショナル型とアンラップを理解しよう」の一部に誤りがあります。

【誤】	● **オプショナル型とは** 値がない状態を許すデータ型は「オプショナル型」で、変数宣言時のクラス名のあとに「?」か「!」を指定します。「?」や「!」を付けない宣言は、値がない状態を許さない「非オプショナル型」になります。一般的なオプショナル型は「?」を利用します。 「?」も「!」は両方とも値がない「nil（ニル）」を保持することが許されています。「?」で宣言したオプショナル変数を利用する場合には、アンラップが必要です。「!」で宣言したオプショナル変数を利用する場合には、アンラップをしなくても使用することができますが、開発者が値があることを保証する必要があります。
【正】	誤:クラス名 正:データ型

P.210：図「geocodeAddressStringメソッドの引数」の内容の一部に誤りがあります

【誤】

```
// 入力された文字から位置情報を取得
geocoder.geocodeAddressString(
    searchKey ,
    completionHandler: { (placemarks, error) in
     （省略）
}) // geocoder ここまで
```

住所やランドマークの文字列から位置情報を取得しています。

▼ geocodeAddressString メソッドの引数

番号	引数名	内容
第一引数	addressString	検索したい場所を記述した文字列。
第二引数	completionHandler	結果と一緒に実行するまとまった処理（クロージャ）。ジオコーダーは、リクエストが成功したか失敗したかに関係なく、このクロージャを実行します。

第一引数であるaddressStringというラベル名は、上のコードでは省略されて書かれています。

【正】

第一引数であるaddressStringの内容は、次の解説が正しいです。

検索したい場所を記述した文字列。**addressString**は省略可能なラベル名であるため、省略せずにコードを書くと「**addressString: searchKey**」となります。今回は省略されているため「**searchKey**」のみコードに書かれています。

P.246：図「ContentView への ZStack の追加と MapView の修正」で追加するコードに誤りがあります。

【誤】

▼ ContentView への ZStack の追加と MapView の修正

```
28                      // 入力が完了したので検索キーワードに設定する
29                      dispSearchKey = inputText
30                  }
31              // 余白を追加
32                  .padding()
33
34              // 奥から手前方向にレイアウト（右下基準で配置する）
35              ZStack(alignment: .bottomTrailing) {
36                  // マップを表示
37                  MapView(searchKey: dispSearchKey, MapType: dispMapType)
38              } // ZStack ここまで
39
40          } // VStack ここまで
```

変更

37行目のコード
MapView(searchKey: dispSearchKey, MapType: dispMapType)

【正】

正しくは、以下のコードを追加します。
MapView(searchKey: dispSearchKey, mapType: dispMapType)

P.264：図「画面遷移を実装」で追加するコメントに誤りがあります。

【誤】

▼ 画面遷移を実装

```
10  struct ContentView: View {
11      var body: some View {
12          NavigationView {
13              VStack {
14                  Text("タイマー画面")
15              } // VStack ここまで
16              // ナビゲーションにボタンを追加
17              .toolbar {
18                  // ナビゲーションバーの左にボタンを追加
19                  ToolbarItem(placement: .navigationBarTrailing) {
20                      // ナビゲーション遷移
```

追加

18行目のコメント
// ナビゲーションバーの左にボタンを追加

【正】

正しくは、以下のコメントを追加します。
// ナビゲーションバーの右にボタンを追加

P.265:「コード解説」のコメントに誤りがあります。

【誤】	``` // ナビゲーションにボタンを追加 .toolbar { // ナビゲーションバーの左にボタンを追加 ToolbarItem(placement: .navigationBarTrailing) { // ナビゲーション遷移 NavigationLink(destination: SettingView()) { // テキストを表示 Text("秒数設定") ``` // ナビゲーションバーの左にボタンを追加
【正】	正しくは、以下のコメントです。 // ナビゲーションバーの右にボタンを追加

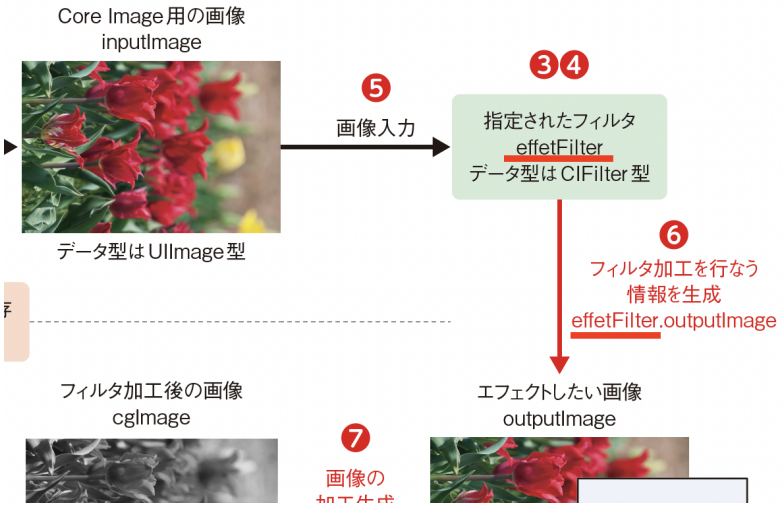
P.288:「コード解説」で一部誤りがあります。

【誤】	タイマーが実行中かどうかチェックします。「if let」を利用して、**timer** をアンラップすることで、**timer** に値がある場合のみ、unwrapedTimerHandler.isValid で、タイマーが実行中かどうかを調べています。
【正】	タイマーが実行中かどうかチェックします。「if let」を利用して、**timerHandler** をアンラップすることで、**timerHandler** に値がある場合（タイマーがスタートしているとき）のみ、unwrapedTimerHandler.isValid で、タイマーが実行中かどうかを調べています。

P.352:章タイトルに誤りがあります。

【誤】	1-2 **PHPickerViewController** ファイルの作成
【正】	1-2 **PHPickerView** ファイルの作成

P.382：図「エフェクト処理イメージ図」の解説に誤りがあります。

【誤】	
【正】	❸❹❻で記述されている**effetFilter**は、**effectFilter**が正しいです。

P.383：Pointの参照先ページ番号に誤りがあります。

【誤】	「CIFilter(name:)」で指定できるフィルタ名の一覧は、「2 フィルタの効果を確認しよう」（**P.402**）で掲載しています。
【正】	「CIFilter(name:)」で指定できるフィルタ名の一覧は、「2 フィルタの効果を確認しよう」（**P.404**）で掲載しています。

P.386：図「createCGImage メソッドの引数」の引数名に誤りがあります。

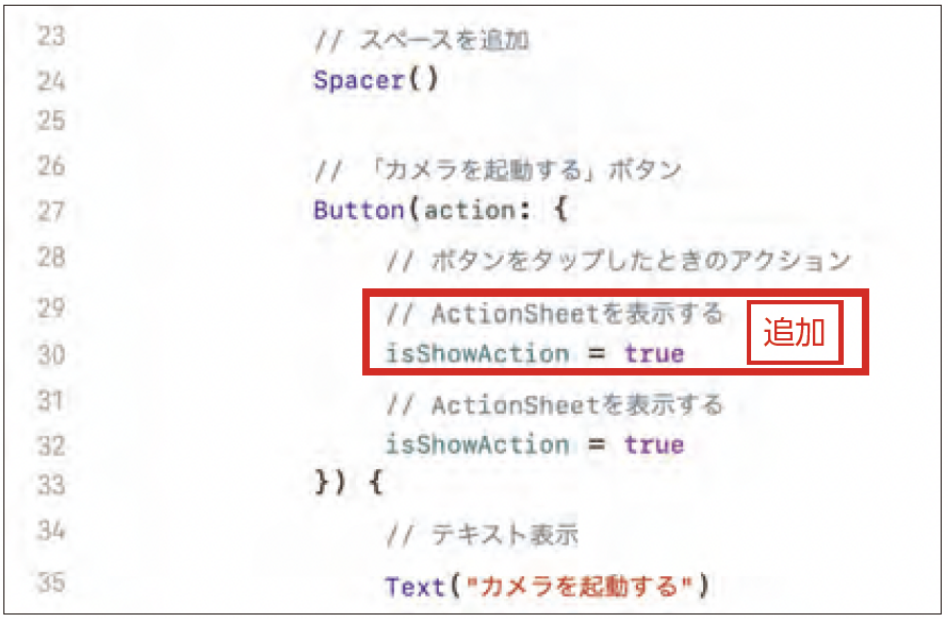
【誤】

▼ createCGImage メソッドの引数

番号	引数名	内容
第一引数	image	CIImage 型（CIImage クラス）の画像をセットします。Core Image の画像オブジェクトです
第二引数	fromRect	レンダリングする画像の領域

【正】 第二引数の引数名**fromRect**は**from**が正しいです。

P.391：図「写真を保存する変数の初期化」で追加するコードに誤りがあります。

【誤】	▼ 写真を保存する変数の初期化  上記で追加されている、以下のコードに誤りがあります。 // ActionSheetを表示する isShowAction = true
【正】	正しくは、以下のコードを追加してください。 // 撮影写真を初期化する captureImage = nil

P.399：参照先ページ番号に誤りがあります。

【誤】	3-1 「3-2「 SNS に投稿する」ボタンの作成」（**P.345**） 3-2 「2-2「 シェア」ボタンのアクションを作る」（**P.386**）
【正】	3-1 「3-2「 SNS に投稿する」ボタンの作成」（**P.346**） 3-2 「2-2「 シェア」ボタンのアクションを作る」（**P.387**）

P.426：ページの下から四行目に誤りがあります。

【誤】	onSubmit モディファイアのとしての処理はTask を起動して終了します。
【正】	onSubmit モディファイアの処理はTask を起動して終了します。

P.450：図「お菓子データを配列へ格納」で追加するコードに一部誤りがあります。

【誤】

「お菓子データを配列へ格納」で追加するコードでは、print文で正しく情報が出力されません。

▼ お菓子データを配列へ格納

```
63              // 受け取ったJSONデータをパース（解析）して格納
64              let json = try decoder.decode(ResultJson.self, from: data)
65
66              // print(json)                                          コメントアウト
67
68              // お菓子の情報が取得できているか確認                                  追加
69              guard let items = json.item else { return }
70              // @Publishedの変数を更新するときはメインスレッドで更新する必要がある
71              DispatchQueue.main.async {
72                  // お菓子のリストを初期化
73                  self.okashiList.removeAll()
74              }
75              // 取得しているお菓子の数だけ処理
76              for item in items {
77                  // お菓子の名称、掲載URL、画像URLをアンラップ
78                  if let name = item.name,
79                     let link = item.url,
80                     let image = item.image {
81                      // 1つのお菓子を構造体でまとめて管理
82                      let okashi = OkashiItem(name: name, link: link, image: image)
83                      // @Publishedの変数を更新するときはメインスレッドで更新する必要がある
84                      DispatchQueue.main.async {
85                          // お菓子の配列へ追加
86                          self.okashiList.append(okashi)
87                      }
88                  }
89              }
90              print(self.okashiList)
91
92          } catch {
```

【正】

print文で正常な出力を確認するためには、次のようにコードを追加してください。

```
63              // 受け取ったJSONデータをパース（解析）して格納
64              let json = try decoder.decode(ResultJson.self, from: data)
65
66              // print(json)                                          コメントアウト
67
68              // お菓子の情報が取得できているか確認                                  追加
69              guard let items = json.item else { return }
70              // @Publishedの変数を更新するときはメインスレッドで更新する必要がある
71              DispatchQueue.main.async {
72                  // お菓子のリストを初期化
73                  self.okashiList.removeAll()
74
75                  // 取得しているお菓子の数だけ処理
76                  for item in items {
77                      // お菓子の名称、掲載URL、画像URLをアンラップ
78                      if let name = item.name,
79                         let link = item.url,
80                         let image = item.image {
81                          // 1つのお菓子を構造体でまとめて管理
82                          let okashi = OkashiItem(name: name, link: link, image: image)
83                          // お菓子の配列へ追加
84                          self.okashiList.append(okashi)
85                      }
86                  }
87                  print(self.okashiList)
88              }
89
90          } catch {
```

P451：コード解説で解説が抜けてしました。

【誤】

DispatchQueue.main.asyncを使用していますが、解説が抜けていました。

```
// お菓子の情報が取得できているか確認
guard let items = json.item  else { return }
```

「json.item」の中に検索結果のお菓子データが格納されています。guard let 文でお菓子データが存在するときに、items にコピーして次の処理を行います。

DispatchQueue.main.asyncの解説を追加します。

```
// お菓子のリストを初期化
self.okashiList.removeAll()
```

okashiList に格納されている全ての要素を削除して、元の状態（初期化）に戻しています。
1 度目の検索の場合は要素を保持していませんが、2 度目以降は前のお菓子情報が格納されているので、okashiList を初期化しています。

【正】

上図の箇所に、以下の解説を追加いたします。

```
// @Publishedの変数を更新するときにはメインスレッドで更新する必要がある
DispatchQueue.main.async {
 …
}
```

DispatchQueue（ディスパッチキュー）は、アプリのメインスレッドやバックグラウンドスレッド上で、タスクの連続または同時実行を管理するオブジェクトです。DispatchQueueは、タスクをFIFOキュー（タスクを入ってきた順番に並べて、入ってきた順に実行する方法）で処理します。「DispatchQueue.main.async { … }」と記述することで、「{ … }」の中のタスクを、メインスレッドで非同期にて実行します。
okashiListは@Publishedのプロパティラッパーの変数として宣言されています。@Publishedの変数を更新するときはメインスレッドで更新する必要があります。

DispatchQueueについて、更に詳しく学習したい方は、次のページを参考にしてください。
Swiftでの非同期処理GDP｜ディスパッチキューの解説
https://ticklecode.com/swfitgdp/

■改善内容：

改善内容は正誤ではありません。誤解を招く記述やわかりにくい箇所の補足をさせて頂きます。

P.179：赤枠のコードの修正箇所がわかりにくいため補足します。

改善前

```
18      func cymbalPlay() {
19          do {
20              // シンバル用のプレイヤーに、音源データを指定
21              cymbalPlayer = try AVAudioPlayer(data: cymbalData)
22
23              // シンバルの音源再生
24              cymbalPlayer.play()
25          } catch {
26              print("シンバルで、エラーが発生しました！")
27          }
28      } // cymbalPlay ここまで
```

修正

改善後

```
11  class SoundPlayer: NSObject {
12      // シンバルの音源データを読み込み
13      let cymbalData = NSDataAsset(name: "cymbalSound")!.data
14
15      // シンバル用プレイヤーの変数
16      var cymbalPlayer = AVAudioPlayer()
17
18      func cymbalPlay() {
19          do {
20              // シンバル用のプレイヤーに、音源データを指定
21              cymbalPlayer = try AVAudioPlayer(data: cymbalData)
22
23              // シンバルの音源再生
24              cymbalPlayer.play()
25          } catch {
26              print("シンバルで、エラーが発生しました！")
27          }
28      } // cymbalPlay ここまで
29
30  }
```

修正

修正箇所がわかりやすいスクリーンショットに改善します。

「func cymbalPlayer(){}」の中で「do {} catch {}」を追加します。