Pythonサイコロゲームで学ぶ！基礎からオブジェクト指向プログラミング（2/3）

この資料は、Pythonプログラミングの初学者を対象とし、サイコロゲームを題材に基礎からオブジェクト指向プログラミング（OOP）までを段階的に理解してもらうことを目的としています。 **___や日本語になっている部分を埋めてください**

ステップ8: コンピュータと1回だけ対戦してみよう

これまでは一人でサイコロを振るゲームでしたが、ここからはコンピュータと対戦する機能を追加していきます。 まずは、1回だけコンピュータとサイコロの出目を比べて勝敗を決めるプログラムを作成しましょう。

考えてみよう（コードの穴埋め）：

```
import random

# サイコロを1個振って、出た目を返す関数
def roll_die()->int:
    return random.randint(1, 6)

# 「もう一度振りますか？」と聞いて、”y”ならTrue、”n”ならFalseを返す関数
def ask_and_check_if_continue()->bool:
    while True:
        y_or_n = input(”もう一度振りますか？ (y/n): ”)
        if y_or_n == ”y”:
            return True
        elif y_or_n == ”n”:
            return False
        else:
            print(”無効な入力です。’y’ または ’n’ を入力してください。”)

# 出た目の履歴リストを受け取って、見やすく表示する関数
def print_history(history_list):
    print(”---これまでの出た目---”)
    for i, roll in enumerate(history_list, start=1):
        print(f”{i}回目: {roll}”)
    print(”--------------------”)
    print(f”合計: {sum(history_list)}”)

# サイコロゲームのソロプレイを実行する関数
```

```
def solo_play():
    roll_history_list = []
    while True:
        roll = roll_die()
        roll_history_list.append(roll)
        print(f"出た目は {roll} です。")
        if not ask_and_check_if_continue():
            print("ゲームを終了します。")
            break
    print_history(roll_history_list)

# コンピュータと1回だけ対戦する関数
def one_match():
    # プレイヤーのサイコロを振る
    player_die = ___
    # コンピュータのサイコロを振る
    computer_die = ___

    # それぞれの出目を表示する
    print(f"あなたの出た目は {player_die} です。コンピュータの出た目は {computer_die} です。")

    # プレイヤーの出目とコンピュータの出目を比較して勝敗を判定する
    もしプレイヤーが勝ちの場合:
        # プレイヤーの勝ちの場合のメッセージを表示
        print("あなたの勝ち！")
    もしプレイヤーが負け、コンピュータが勝った場合:
        # コンピュータの勝ちの場合のメッセージを表示
        print("コンピュータの勝ち！")
    else:
        # 引き分けの場合のメッセージを表示
        print("引き分け！")

# このスクリプトが直接実行されたときに、one_match関数を呼び出す
if __name__ == "__main__":
    # one_match関数を実行する
    ___
```

ヒント:

- プレイヤーとコンピュータ、それぞれサイコロを振る処理が必要です。
- `if`、`elif`、`else` を使って、出目の大小を比較し、結果を表示します。
- 勝ち、負け、引き分けの3つのパターンを考えましょう。

解答例:

```
import random

# サイコロを1個振って、出た目を返す関数
def roll_die():
    return random.randint(1, 6)
```

```
# 「もう一度振りますか？」と聞いて、"y"ならTrue、"n"ならFalseを返す関数
def ask_and_check_if_continue():
    while True:
        y_or_n = input("もう一度振りますか？ (y/n): ")
        if y_or_n == "y":
            return True
        elif y_or_n == "n":
            return False
        else:
            print("無効な入力です。'y' または 'n' を入力してください。")

# 出た目の履歴リストを受け取って、見やすく表示する関数
def print_history(history_list):
    print("---これまでの出た目---")
    for i, roll in enumerate(history_list, start=1):
        print(f"{i}回目: {roll}")
    print("--------------------")
    print(f"合計: {sum(history_list)}")

# サイコロゲームのソロプレイを実行する関数
def solo_play():
    roll_history_list = []
    while True:
        roll = roll_die()
        roll_history_list.append(roll)
        print(f"出た目は {roll} です。")
        if not ask_and_check_if_continue():
            print("ゲームを終了します。")
            break
    print_history(roll_history_list)

# コンピュータと1回だけ対戦する関数
def one_match():
    # プレイヤーのサイコロを振る
    player_die = roll_die()
    # コンピュータのサイコロを振る
    computer_die = roll_die()

    # それぞれの出目を表示する
    print(f"あなたの出た目は {player_die} です。コンピュータの出た目は {computer_die} です。")

    # プレイヤーの出目とコンピュータの出目を比較して勝敗を判定する
    if player_die > computer_die:
        # プレイヤーの勝ちの場合のメッセージを表示
        print("あなたの勝ち！")
    elif player_die < computer_die:
        # コンピュータの勝ちの場合のメッセージを表示
        print("コンピュータの勝ち！")
    else:
        # 引き分けの場合のメッセージを表示
        print("引き分け！")

# このスクリプトが直接実行されたときに、one_match関数を呼び出す
if __name__ == "__main__":
```

```
# one_match関数を実行する
one_match()
```

ポイント：

- 新しく `one_match` という関数を作成し、その中で対戦処理を記述しました。
- プレイヤーとコンピュータがそれぞれ `roll_die()` 関数を使ってサイコロを振ります。
- **条件分岐（`if`, `elif`, `else`）** を使って、出目の結果に応じたメッセージを表示します。
- `if __name__ == "__main__":` のブロックで `one_match()` を呼び出すことで、スクリプト実行時に一回対戦が行われます。

> コラム 今回はコンピュータもプレイヤーも同じ `roll_die()` 関数を使っています。もし将来的に「コンピュータは少しイカサマをする」といった特別なルールを加えたくなった場合、コンピュータ専用のサイコロを振る関数を別途作成することも考えられますね。

ステップ9: コンピュータと連続対戦できるようにしよう

これまでのステップでは、1回だけコンピュータと対戦する機能を作成しました。 このステップでは、複数回対戦できるように機能を拡張します。

考えてみよう（コードの穴埋め）：

```
import random

# サイコロを振って目を返す
def roll_die(sides=6):
    return random.randint(1, sides)

# 続行確認
def ask_and_check_if_continue(prompt="もう一度振りますか？ (y/n): ")-> bool:
    while True:
        ans = input(prompt).lower()
        if ans in ("y", "n"):
            return ans == "y"
        print("無効な入力です。'y' または 'n' を入力してください。")

def print_history(history):
    ...

def solo_play():
```

```
    ...

def one_match()->[int,int, str]:
    """コンピュータとの対戦"""
    player_die = roll_die()
    computer_die = roll_die()
    print(f"あなたの出た目は {player_die} です。コンピュータの出た目は {computer_die} です。")
    winner = ""
    if player_die > computer_die:
        print("あなたの勝ち！")
        winner = "player"
    elif player_die < computer_die:
        print("コンピュータの勝ち！")
        winner = "computer"
    else:
        print("引き分け！")
        winner = "draw"
    return ___, ___, ___

# コンピュータとの対戦
def play_vs_computer():

    player_win_count = 0
    computer_win_count = 0
    draw_count = 0

    while True:
        player_die, computer_die, winner = ___()

        if winner == "player":
            ___
        elif: winner =="computer"":
            ___
        else:
            ___

        if not ask_and_check_if_continue("もう一度対戦しますか？ (y/n): "):
            print("対戦を終了します。")
            break

    print("対戦結果:")
    print("あなたの勝利数:", player_win_count)
    print("コンピュータの勝利数:", computer_win_count)
    print("引き分け数:", draw_count)


# メイン処理
if __name__ == "__main__":
    print("モードを選択してください:¥n 1: ソロプレイ¥n 2: コンピュータと対戦")
    mode = input("選択 (1/2): ")
    if mode == "1":
        ___()
    elif mode == "2":
        ___()
```

```
    else:
        print(”無効な選択です。”)
```

ヒント:

- 対戦結果を記録するために、勝利数や引き分け数をカウントする変数を用意します。
- `while True:` ループを使って、ユーザーが続ける限り対戦を繰り返します。

解答例:

```
import random

# サイコロを振って目を返す
def roll_die(sides=6):
    return random.randint(1, sides)

# 続行確認
def ask_and_check_if_continue(prompt=”もう一度振りますか？ (y/n): ”)-> bool:
    while True:
        ans = input(prompt).lower()
        if ans in (”y”, ”n”):
            return ans == ”y”
        print(”無効な入力です。’y’ または ’n’ を入力してください。”)

# 出た目の履歴リストを受け取って、見やすく表示する関数
def print_history(history_list):
    print(”---これまでの出た目---”)
    for i, roll in enumerate(history_list, start=1):
        print(f”{i}回目: {roll}”)
    print(”--------------------”)
    print(f”合計: {sum(history_list)}”)

# サイコロゲームのソロプレイを実行する関数
def solo_play():
    roll_history_list = []
    while True:
        roll = roll_die()
        roll_history_list.append(roll)
        print(f”出た目は {roll} です。”)
        if not ask_and_check_if_continue():
            print(”ゲームを終了します。”)
            break
    print_history(roll_history_list)


def one_match()->[int, int, str]:
    ”””コンピュータとの対戦”””
    player_die = roll_die()
    computer_die = roll_die()
    print(f”あなたの出た目は {player_die} です。コンピュータの出た目は {computer_die} です。”)
    winner = ””
```

```
    if player_die > computer_die:
        print("あなたの勝ち！")
        winner = "player"
    elif player_die < computer_die:
        print("コンピュータの勝ち！")
        winner = "computer"
    else:
        print("引き分け！")
        winner = "draw"
    return player_die, computer_die, winner

# コンピュータとの対戦
def play_vs_computer():

    player_win_count = 0
    computer_win_count = 0
    draw_count = 0

    while True:
        player_die, computer_die, winner = one_match()

        if winner == "player":
            player_win_count += 1
        elif winner == "computer":
            computer_win_count += 1
        else:
            draw_count += 1

        if not ask_and_check_if_continue("もう一度対戦しますか？ (y/n): "):
            print("対戦を終了します。")
            break

    print("対戦結果:")
    print("あなたの勝利数:", player_win_count)
    print("コンピュータの勝利数:", computer_win_count)
    print("引き分け数:", draw_count)


# メイン処理
if __name__ == "__main__":
    print("モードを選択してください:¥n 1: ソロプレイ¥n 2: コンピュータと対戦")
    mode = input("選択 (1/2): ")
    if mode == "1":
        solo_play()
    elif mode == "2":
        play_vs_computer()
    else:
        print("無効な選択です。")
```

ポイント:

- **対戦結果のカウント**: プレイヤーの勝利数、コンピュータの勝利数、引き分け数をそれぞれカウントする変数を用意しました。

- `while True:` **ループ**: ユーザーが続ける限り対戦を繰り返すために使用します。`ask_and_check_if_continue()` 関数でユーザーの入力を確認し、続行するかどうかを判断します。
- **関数の再利用**: `one_match()` 関数を使って、対戦のロジックをまとめました。これにより、コードがスッキリし、再利用性が高まります。

ステップ10: リストを使って対戦履歴をもっと充実させよう

ステップ9では対戦の勝敗数を記録しました。 このステップでは、リストをより効果的に活用して、対戦履歴をもっと詳細に記録し、統計情報も表示できるようにしましょう。

考えてみよう（コードの穴埋め）:

```
import random

# サイコロを振って目を返す
def roll_die(sides=6):
    return random.randint(1, sides)

# 続行確認
def ask_and_check_if_continue(prompt="もう一度振りますか？ (y/n): "):
    while True:
        ans = input(prompt).lower()
        if ans in ("y", "n"):
            return ans == "y"
        print("無効な入力です。'y' または 'n' を入力してください。")

# 出た目の履歴リストを受け取って、見やすく表示する関数
def print_history(history_list):
    print("---これまでの出た目---")
    for i, roll in enumerate(history_list, start=1):
        print(f"{i}回目: {roll}")
    print("--------------------")
    print(f"合計: {sum(history_list)}")

# サイコロゲームのソロプレイを実行する関数
def solo_play():
    roll_history_list = []
    while True:
        roll = roll_die()
        roll_history_list.append(roll)
        print(f"出た目は {roll} です。")
        if not ask_and_check_if_continue():
            print("ゲームを終了します。")
            break
    print_history(roll_history_list)
```

```
def one_match():
    """コンピュータとの対戦"""
    player_die = roll_die()
    computer_die = roll_die()
    print(f"あなたの出た目は {player_die} です。コンピュータの出た目は {computer_die} です。")

    winner = ""
    if player_die > computer_die:
        print("あなたの勝ち！")
        winner = "player"
    elif player_die < computer_die:
        print("コンピュータの勝ち！")
        winner = "computer"
    else:
        print("引き分け！")
        winner = "draw"

    return player_die, computer_die, winner

# 対戦履歴の詳細統計を表示する関数
def print_match_statistics(match_history:list[int,int,str]):
    """
    match_history: 各要素が［プレイヤー出目, コンピュータ出目, 勝者］のリスト
    """
    if not match_history:
        print("対戦履歴がありません。")
        return

    print("¥n---詳細な対戦履歴---")

    for i, match in ___(match_history, start=1): # iには0,1,2,3... matchにはmatch_historyの一つの要素である、
[player_dice, computer_dice, winner]のリストが入る
        player_roll, computer_roll, result = ___ # matchの要素を取り出す
        print(f"{i}戦目: あなた:{player_roll}, コンピュータ:{computer_roll}, 結果:{result}")

    print("¥n---統計情報---")

    # プレイヤーの出目だけを抽出してリストにする
    player_rolls = ___  # match_historyから各matchの0番目の要素（プレイヤーの出目）を取り出してリストにする
    # コンピュータの出目だけを抽出してリストにする
    computer_rolls = ___  # match_historyから各matchの1番目の要素（コンピュータの出目）を取り出してリストにする

    # プレイヤーの統計情報を表示
    print(f"あなたの出目統計:")
    print(f"  最高出目: {___(player_rolls)}")
    print(f"  最低出目: {___(player_rolls)}")
    print(f"  平均出目: {sum(player_rolls) / ___(player_rolls):.2f}")

    # コンピュータの統計情報を表示
    print(f"コンピュータの出目統計:")
    print(f"  最高出目: {___(computer_rolls)}")
    print(f"  最低出目: {___(computer_rolls)}")
    print(f"  平均出目: {sum(computer_rolls) / ___(computer_rolls):.2f}")
```

```
    # 勝敗数を集計
    player_win_count = 0
    computer_win_count = 0
    draw_count = 0

    # match_historyの各要素をループして勝敗数をカウント
    for match in match_history:
        _, _, winner = match
        if winner == "player":
            ___  # player_win_countを1増やす
        elif winner == "computer":
            ___  # computer_win_countを1増やす
        else:
            ___  # draw_countを1増やす

    print(f"¥n最終結果:")
    print(f"  あなたの勝利: {player_win_count}回")
    print(f"  コンピュータの勝利: {computer_win_count}回")
    print(f"  引き分け: {draw_count}回")

# コンピュータとの対戦（履歴記録機能付き）
def play_vs_computer():
    match_history = []  # 各対戦の詳細情報を保存するリスト

    while True:
        player_die, computer_die, winner = one_match()

        # 対戦結果を詳細履歴に追加（リストの形で）
        ___  # match_historyに [player_die, computer_die, winner] を追加

        if not ask_and_check_if_continue("もう一度対戦しますか？ (y/n): "):
            print("対戦を終了します。")
            break

    # 詳細統計を表示
    ___))

# メイン処理
if __name__ == "__main__":
    print("モードを選択してください:¥n 1: ソロプレイ¥n 2: コンピュータと対戦")
    mode = input("選択 (1/2): ")
    if mode == "1":
        solo_play()
    elif mode == "2":
        play_vs_computer()
    else:
        print("無効な選択です。")
```

ヒント:

- `enumerate` を使って、リストのインデックスと要素を同時に取得できます。
- リスト内包表記を使って、特定の要素だけを抽出できます。例：`[match[0] for match in match_history]`

- `max()`, `min()`, `sum()`, `len()` 関数を使って統計情報を計算できます。
- 各対戦の情報を `[プレイヤー出目, コンピュータ出目, 勝者]` の形のリストとして保存します。
- リストの `append()` メソッドを使って、新しい対戦結果を履歴に追加します。

解答例:

```
import random

# サイコロを振って目を返す
def roll_die(sides=6):
    return random.randint(1, sides)

# 続行確認
def ask_and_check_if_continue(prompt="もう一度振りますか？ (y/n): "):
    while True:
        ans = input(prompt).lower()
        if ans in ("y", "n"):
            return ans == "y"
        print("無効な入力です。'y' または 'n' を入力してください。")

# 出た目の履歴リストを受け取って、見やすく表示する関数
def print_history(history_list):
    print("---これまでの出た目---")
    for i, roll in enumerate(history_list, start=1):
        print(f"{i}回目: {roll}")
    print("--------------------")
    print(f"合計: {sum(history_list)}")

# サイコロゲームのソロプレイを実行する関数
def solo_play():
    roll_history_list = []
    while True:
        roll = roll_die()
        roll_history_list.append(roll)
        print(f"出た目は {roll} です。")
        if not ask_and_check_if_continue():
            print("ゲームを終了します。")
            break
    print_history(roll_history_list)

def one_match():
    """コンピュータとの対戦"""
    player_die = roll_die()
    computer_die = roll_die()
    print(f"あなたの出た目は {player_die} です。コンピュータの出た目は {computer_die} です。")

    winner = ""
    if player_die > computer_die:
        print("あなたの勝ち！")
        winner = "player"
    elif player_die < computer_die:
```

```
        print(”コンピュータの勝ち！”)
        winner = ”computer”
    else:
        print(”引き分け！”)
        winner = ”draw”

    return player_die, computer_die, winner

# 対戦履歴の詳細統計を表示する関数
def print_match_statistics(match_history):
    ”””
    match_history: 各要素が［プレイヤー出目, コンピュータ出目, 勝者］のリスト
    ”””
    if not match_history:
        print(”対戦履歴がありません。”)
        return

    print(”¥n---詳細な対戦履歴---”)
    # enumerateを使って各対戦の詳細を表示
    for i, match in enumerate(match_history, start=1):
        player_roll, computer_roll, result = match
        print(f”{i}戦目: あなた:{player_roll}, コンピュータ:{computer_roll}, 結果:{result}”)

    print(”¥n---統計情報---”)

    # プレイヤーの出目だけを抽出してリストにする
    player_rolls = [match[0] for match in match_history]
    # コンピュータの出目だけを抽出してリストにする
    computer_rolls = [match[1] for match in match_history]

    # プレイヤーの統計情報を表示
    print(f”あなたの出目統計:”)
    print(f”  最高出目: {max(player_rolls)}”)
    print(f”  最低出目: {min(player_rolls)}”)
    print(f”  平均出目: {sum(player_rolls) / len(player_rolls):.2f}”)

    # コンピュータの統計情報を表示
    print(f”コンピュータの出目統計:”)
    print(f”  最高出目: {max(computer_rolls)}”)
    print(f”  最低出目: {min(computer_rolls)}”)
    print(f”  平均出目: {sum(computer_rolls) / len(computer_rolls):.2f}”)

    # 勝敗数を集計
    player_win_count = 0
    computer_win_count = 0
    draw_count = 0

    # match_historyの各要素をループして勝敗数をカウント
    for match in match_history:
        _, _, winner = match
        if winner == ”player”:
            player_win_count += 1
        elif winner == ”computer”:
            computer_win_count += 1
        else:
```

```
            draw_count += 1

    print(f"¥n最終結果:")
    print(f"  あなたの勝利: {player_win_count}回")
    print(f"  コンピュータの勝利: {computer_win_count}回")
    print(f"  引き分け: {draw_count}回")

# コンピュータとの対戦（履歴記録機能付き）
def play_vs_computer():
    match_history = []  # 各対戦の詳細情報を保存するリスト

    while True:
        player_die, computer_die, winner = one_match()

        # 対戦結果を詳細履歴に追加（リストの形で）
        match_history.append([player_die, computer_die, winner])

        if not ask_and_check_if_continue("もう一度対戦しますか？ (y/n): "):
            print("対戦を終了します。")
            break

    # 詳細統計を表示
    print_match_statistics(match_history)

# メイン処理
if __name__ == "__main__":
    print("モードを選択してください:¥n 1: ソロプレイ¥n 2: コンピュータと対戦")
    mode = input("選択 (1/2): ")
    if mode == "1":
        solo_play()
    elif mode == "2":
        play_vs_computer()
    else:
        print("無効な選択です。")
```

ポイント:

- **`enumerate`の活用**: `enumerate` を使うことで、リストのインデックスと要素を同時に取得できます。これにより、対戦履歴を表示する際に、何戦目かを簡単に表示できます。
- **リスト内包表記**: `[match[0] for match in match_history]` のように書くことで、リストの各要素から特定の部分だけを抽出して新しいリストを作成できます。これは非常に便利でPythonらしい書き方です。
- **ネストしたリスト**: `match_history` は「リストのリスト」という構造になっています。各対戦の情報を `[プレイヤー出目, コンピュータ出目, 勝者]` の形で保存し、それらをまとめて一つの大きなリストで管理します。
- **統計関数の活用**: `max()`, `min()`, `sum()`, `len()` といったPython標準の関数を使って、簡単に統計情報を計算できます。
- **関数の分離**: 統計表示機能を `print_match_statistics()` という独立した関数にすることで、コードの見通しが良くなり、テストもしやすくなります。

ステップ11：辞書（dict）を使ってゲーム設定を一元管理しよう

これまでのプログラムでは、サイコロの面数（6）やメッセージ（”あなたの勝ち！”など）がコード内に直接書かれていました。 このステップでは、**辞書（dictionary）**を使ってこれらの設定を一元管理し、より保守しやすいコードに改良しましょう。

考えてみよう（コードの穴埋め）：

```
import random

# 結果の定数を定義
PLAYER_WIN = ”player”
COMPUTER_WIN = ”computer”
DRAW = ”draw”

# ゲーム設定を辞書で一元管理
game_config = {
    ”sides”: ___,  # サイコロの面数
    ”messages”: {
        ___: ”あなたの勝ち！”,
        ___: ”コンピュータの勝ち！”,
        ___: ”引き分け！”
    }
}

# サイコロを振って目を返す（面数は設定から取得）
def roll_die(sides=None):
    if sides is None:
        # game_configから面数を取得
        sides = ___
    return random.randint(1, sides)

# 続行確認
def ask_and_check_if_continue(prompt=”もう一度振りますか？ (y/n): ”):
    while True:
        ans = input(prompt).lower()
        if ans in (”y”, ”n”):
            return ans == ”y”
        print(”無効な入力です。’y’ または ’n’ を入力してください。”)

# 出た目の履歴リストを受け取って、見やすく表示する関数
def print_history(history_list):
    print(”---これまでの出た目---”)
    for i, roll in enumerate(history_list, start=1):
        print(f”{i}回目: {roll}”)
    print(”--------------------”)
    print(f”合計: {sum(history_list)}”)
```

```
# サイコロゲームのソロプレイを実行する関数
def solo_play():
    roll_history_list = []
    while True:
        roll = roll_die()
        roll_history_list.append(roll)
        print(f"出た目は {roll} です。")
        if not ask_and_check_if_continue():
            print("ゲームを終了します。")
            break
    print_history(roll_history_list)

# 一回の対戦を実行して結果を返す
def one_match():
    player_die = roll_die()
    computer_die = roll_die()
    print(f"あなたの出た目は {player_die} です。コンピュータの出た目は {computer_die} です。")

    if player_die > computer_die:
        result = ___  # PLAYER_WINを設定
    elif player_die < computer_die:
        result = ___  # COMPUTER_WINを設定
    else:
        result = ___  # DRAWを設定

    # 結果メッセージをgame_configから取得して表示
    message = ___
    print(message)

    return player_die, computer_die, result

# 対戦履歴の詳細統計を表示する関数
def print_match_statistics(match_history):
    if not match_history:
        print("対戦履歴がありません。")
        return

    print("¥n---詳細な対戦履歴---")
    for i, match in enumerate(match_history, start=1):
        player_roll, computer_roll, result = match
        print(f"{i}戦目: あなた:{player_roll}, コンピュータ:{computer_roll}, 結果:{result}")

    print("¥n---統計情報---")

    player_rolls = [match[0] for match in match_history]
    computer_rolls = [match[1] for match in match_history]

    print(f"あなたの出目統計:")
    print(f"  最高出目: {max(player_rolls)}")
    print(f"  最低出目: {min(player_rolls)}")
    print(f"  平均出目: {sum(player_rolls) / len(player_rolls):.2f}")

    print(f"コンピュータの出目統計:")
    print(f"  最高出目: {max(computer_rolls)}")
    print(f"  最低出目: {min(computer_rolls)}")
```

```
    print(f"  平均出目: {sum(computer_rolls) / len(computer_rolls):.2f}")

    # 勝敗数を辞書で管理
    win_counts = {___: 0, ___: 0, ___: 0}  # PLAYER_WIN, COMPUTER_WIN, DRAWのキーで初期化

    for match in match_history:
        winner = match[2]
        ___  # win_counts[winner] を1増やす

    print(f"¥n最終結果:")
    print(f"  あなたの勝利: {win_counts[___]}回")      # PLAYER_WINのカウントを表示
    print(f"  コンピュータの勝利: {win_counts[___]}回")  # COMPUTER_WINのカウントを表示
    print(f"  引き分け: {win_counts[___]}回")          # DRAWのカウントを表示

# コンピュータとの対戦（辞書による設定管理版）
def play_vs_computer():
    match_history = []

    while True:
        player_die, computer_die, winner = one_match()
        match_history.append([player_die, computer_die, winner])

        if not ask_and_check_if_continue("もう一度対戦しますか？ (y/n): "):
            print("対戦を終了します。")
            break

    print_match_statistics(match_history)

# メイン処理
if __name__ == "__main__":
    print("モードを選択してください:¥n 1: ソロプレイ¥n 2: コンピュータと対戦")
    mode = input("選択 (1/2): ")
    if mode == "1":
        solo_play()
    elif mode == "2":
        play_vs_computer()
    else:
        print("無効な選択です。")
```

ヒント:

- 定数 `PLAYER_WIN`, `COMPUTER_WIN`, `DRAW` を辞書のキーとして使用します。
- `game_config["messages"][結果]` でメッセージを取得できます。
- `game_config["sides"]` でサイコロの面数を取得できます。
- 辞書の初期化では、各キーに対して値0を設定します。
- `辞書名[キー] += 1` で辞書の値を増やすことができます。

解答例:

```
import random

# 結果の定数を定義
PLAYER_WIN = "player"
COMPUTER_WIN = "computer"
DRAW = "draw"

# ゲーム設定を辞書で一元管理
game_config = {
    "sides": 6,  # サイコロの面数
    "messages": {
        PLAYER_WIN: "あなたの勝ち！",
        COMPUTER_WIN: "コンピュータの勝ち！",
        DRAW: "引き分け！"
    }
}

# サイコロを振って目を返す（面数は設定から取得）
def roll_die(sides=None):
    if sides is None:
        # game_configから面数を取得
        sides = game_config["sides"]
    return random.randint(1, sides)

# 続行確認
def ask_and_check_if_continue(prompt="もう一度振りますか？ (y/n): "):
    while True:
        ans = input(prompt).lower()
        if ans in ("y", "n"):
            return ans == "y"
        print("無効な入力です。'y' または 'n' を入力してください。")

# 出た目の履歴リストを受け取って、見やすく表示する関数
def print_history(history_list):
    print("---これまでの出た目---")
    for i, roll in enumerate(history_list, start=1):
        print(f"{i}回目: {roll}")
    print("--------------------")
    print(f"合計: {sum(history_list)}")

# サイコロゲームのソロプレイを実行する関数
def solo_play():
    roll_history_list = []
    while True:
        roll = roll_die()
        roll_history_list.append(roll)
        print(f"出た目は {roll} です。")
        if not ask_and_check_if_continue():
            print("ゲームを終了します。")
            break
    print_history(roll_history_list)

# 一回の対戦を実行して結果を返す
def one_match():
```

```
    player_die = roll_die()
    computer_die = roll_die()
    print(f"あなたの出た目は {player_die} です。コンピュータの出た目は {computer_die} です。")

    if player_die > computer_die:
        result = PLAYER_WIN
    elif player_die < computer_die:
        result = COMPUTER_WIN
    else:
        result = DRAW

    # 結果メッセージをgame_configから取得して表示
    message = game_config["messages"][result]
    print(message)

    return player_die, computer_die, result

# 対戦履歴の詳細統計を表示する関数
def print_match_statistics(match_history):
    if not match_history:
        print("対戦履歴がありません。")
        return

    print("¥n---詳細な対戦履歴---")
    for i, match in enumerate(match_history, start=1):
        player_roll, computer_roll, result = match
        print(f"{i}戦目: あなた:{player_roll}, コンピュータ:{computer_roll}, 結果:{result}")

    print("¥n---統計情報---")

    player_rolls = [match[0] for match in match_history]
    computer_rolls = [match[1] for match in match_history]

    print(f"あなたの出目統計:")
    print(f"  最高出目: {max(player_rolls)}")
    print(f"  最低出目: {min(player_rolls)}")
    print(f"  平均出目: {sum(player_rolls) / len(player_rolls):.2f}")

    print(f"コンピュータの出目統計:")
    print(f"  最高出目: {max(computer_rolls)}")
    print(f"  最低出目: {min(computer_rolls)}")
    print(f"  平均出目: {sum(computer_rolls) / len(computer_rolls):.2f}")

    # 勝敗数を辞書で管理
    win_counts = {PLAYER_WIN: 0, COMPUTER_WIN: 0, DRAW: 0}

    for match in match_history:
        _, _, winner = match
        win_counts[winner] += 1

    print(f"¥n最終結果:")
    print(f"  あなたの勝利: {win_counts[PLAYER_WIN]}回")
    print(f"  コンピュータの勝利: {win_counts[COMPUTER_WIN]}回")
    print(f"  引き分け: {win_counts[DRAW]}回")
```

```
# コンピュータとの対戦（辞書による設定管理版）
def play_vs_computer():
    match_history = []

    while True:
        player_die, computer_die, winner = one_match()
        match_history.append([player_die, computer_die, winner])

        if not ask_and_check_if_continue("もう一度対戦しますか？ (y/n): "):
            print("対戦を終了します。")
            break

    print_match_statistics(match_history)

# メイン処理
if __name__ == "__main__":
    print("モードを選択してください:¥n 1: ソロプレイ¥n 2: コンピュータと対戦")
    mode = input("選択 (1/2): ")
    if mode == "1":
        solo_play()
    elif mode == "2":
        play_vs_computer()
    else:
        print("無効な選択です。")
```

ポイント:

- **定数の活用**: `PLAYER_WIN`, `COMPUTER_WIN`, `DRAW` として文字列リテラルを定数化し、タイプミスを防ぎ、保守性を向上させました。**マジックナンバー**とも呼ばれる重要な概念
- **辞書による設定管理**: `game_config` 辞書を使って、サイコロの面数やメッセージを一元管理しています。
- **ネストした辞書**: `game_config["messages"]` は辞書の中に辞書を持つ構造になっています。
- **マジックナンバーの排除**: 直接 `6` を書く代わりに `game_config["sides"]` を使用しています。
- **辞書によるカウント**: `win_counts` 辞書を使って、各結果の回数を効率的に管理しています。
- **拡張性**: 設定を変更したい場合は、`game_config` を修正するだけで済みます。

> コラム 設定を変更してみよう！

```
# 10面サイコロに変更
game_config["sides"] = 10

# メッセージをカスタマイズ
game_config["messages"][PLAYER_WIN] = "やったね！君の勝ちだ！"
```

このように、辞書を使うことで、コードの他の部分を変更することなく、簡単に設定を変更できます。これが「設定の一元管理」の威力です！

コラム **zip関数での履歴表示**

2つのリストを同時にループしたい場合、`zip()` 関数が便利です：

```
# 従来の方法
for i in range(len(player_history)):
    print(f"{i+1}回戦: あなた {player_history[i]} - コンピュータ {comp_history[i]}")

# zipとenumerateを組み合わせた方法
for i, (p, c) in enumerate(zip(player_history, comp_history), start=1):
    print(f"{i}回戦: あなた {p} - コンピュータ {c}")
```

より簡潔で読みやすいコードになります！