Pythonプログラミング補足資料１：基本のキ

この資料は、「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（1/3)」の演習を進める上で、特に最初のステップでつまずきやすいPythonの基本的なルールや用語について解説します。

1. Pythonプログラムの実行方法

Pythonのコードを実行するには、いくつかの方法があります。

- **Pythonインタプリタの対話モード:** ターミナル（WindowsではコマンドプロンプトやPowerShell）で `python` と入力すると起動します。1行ずつコードを入力してすぐに結果を確認できます。ちょっとした動作確認に便利です。

```
python
>>> print(”こんにちは”)
こんにちは
>>> 1 + 2
3
>>> exit()
```

- **スクリプトファイルとして実行:** テキストエディタ（VS Codeなど）で `.py` という拡張子のファイルにコードを保存し、ターミナルで `python ファイル名.py` と入力して実行します。演習では主にこの方法を使います。

```
python myscript.py
```

VS Codeを使っている場合は、エディタの実行ボタン（通常は右上の再生マークのようなアイコン）からも実行できます。

2. Pythonの基本ルール

- **コメント:** コード中に `#` を書くと、その行の `#` 以降の部分はプログラムの動作に影響しない「コメント」になります。メモや説明を残すのに使います。

```
# これはコメントです
x = 10 # これもコメント
```

- **インデント（字下げ）：** Pythonでは、行頭のスペース（インデント）が非常に重要です。`if`文や`while`ループ、関数の定義など、コードのまとまり（ブロック）を表すのに使われます。インデントが正しくないとエラーになります。通常はスペース4つで1つのインデントレベルとします。

```
# 正しい例
if x > 0:
    print("xは正です") # ifブロックの中（インデントされている）
else:
    print("xは0以下です") # elseブロックの中（インデントされている）

# 間違った例（IndentationError）
# if x > 0:
# print("xは正です") # インデントがない
```

3. モジュールと `import`

Pythonには便利な機能がたくさん詰まった「モジュール」という部品があります。 `import モジュール名` と書くことで、そのモジュールの機能を使えるようになります。 演習では `random` モジュールを使い、ランダムな数値を生成します。

```
import random # randomモジュールを使えるようにする

# 1から6までのランダムな整数を生成
dice_roll = random.randint(1, 6)
print(f"サイコロの目は {dice_roll} です")
```

4. 変数とデータ型

- **変数:** 数値や文字列などのデータに名前を付けて保存しておくためのものです。箱のようなイメージです。

```
my_name = "AIプログラマー" # "AIプログラマー" という文字列を my_name という変数に保存
age = 20                  # 20 という数値を age という変数に保存
```

- **データ型:** 変数が扱うデータの種類のことです。Pythonは自動で型を判別してくれますが、主なものを覚えておきましょう。
  - **整数（`int`）:** `1`, `100`, `-5` のような小数点のない数値。

- **浮動小数点数（`float`）:** `3.14`, `-0.5` のような小数点のある数値。
- **文字列（`str`）:** `¥"こんにちは¥"`, `¥'Python¥'` のように文字の並びを `¥' ¥'` や `¥" ¥"` で囲んだもの。
- **ブール値（`bool`）:** `True`（真）または `False`（偽）のどちらかの値。条件分岐などで使われます。

- **型変換:** あるデータ型を別のデータ型に変換することです。
  - `str(数値)`: 数値を文字列に変換します。 `print()` で数値と文字列を連結する際によく使います。

```
age = 20
print("私の年齢は" + str(age) + "歳です") # str(20) は ¥"20¥" になる
```

  - `int(文字列)`: 文字列を整数に変換します。 `input()` で受け取った文字列を数値として扱いたい場合に使います。

```
num_str = "10"
num_int = int(num_str) # ¥"10¥" が 10 になる
```

5. 基本的な入出力

- **画面への出力（`print()`）:** カッコ `()` の中に指定した値や変数の内容を画面に表示します。

```
print("Hello, Python!")
x = 10
print(x)
print("xの値は", x, "です") # カンマで区切ると半角スペースで連結される
print(f"xの値は {x} です") # f-string（演習資料のコラム参照）
```

- **キーボードからの入力（`input()`）:** ユーザーにキーボードから値を入力してもらい、その値を文字列として受け取ります。

```
user_name = input("お名前を入力してください: ")
print(f"こんにちは、{user_name}さん！")

age_str = input("年齢を入力してください: ")
age_int = int(age_str) # 文字列として受け取るので、数値として使う場合はint()で変換
print(f"{age_int}歳なんですね！")
```

6. 条件分岐（`if` 文）

「もし〇〇ならば△△する、そうでなければ□□する」というように、条件によって処理を変えたい場合に使います。

- **基本構造:**

```
if 条件式1:
    # 条件式1がTrueの場合に実行される処理
elif 条件式2: # ¥"else if¥" の略
    # 条件式1がFalseで、条件式2がTrueの場合に実行される処理
else:
    # すべての条件式がFalseの場合に実行される処理
```

- **比較演算子:** 条件式でよく使われます。
  - `==` : 等しい
  - `!=` : 等しくない
  - `>` : より大きい
  - `<` : より小さい
  - `>=` : 以上
  - `<=` : 以下

```
score = 85
if score >= 80:
    print("合格です！")
elif score >= 60:
    print("追試です。")
else:
    print("不合格です。")
```

7. 繰り返し（`while`ループ）

特定の条件が満たされている間、同じ処理を繰り返したい場合に使います。

- **基本構造:**

```
while 条件式:
    # 条件式がTrueの間、繰り返し実行される処理
```

- **`break`:** ループを途中で強制的に終了します。

- `continue`: ループの現在の回の残りの処理をスキップし、次の回の条件判定に進みます。

```
count = 0
while count < 3:
    print(f"カウント: {count}")
    count += 1 # count = count + 1 と同じ
print("ループ終了")

# ¥'q¥' が入力されるまで入力を促す例
while True: # 無限ループ（breakで抜ける必要がある）
    user_input = input("何か入力してください（¥'q¥'で終了): ")
    if user_input == ¥'q¥':
        print("終了します。")
        break # ループを抜ける
    print(f"入力されたのは: {user_input} です。")
```