

# 第4回

## 関数とオブジェクト(メソッド)

### 目次

- 関数とは
- 関数を定義する
- 関数を呼び出す
- 引数のキーワード指定とデフォルト値
- 複数の戻り値
- オブジェクト指向とは
- Pythonにおけるオブジェクト
- クラス
- dateオブジェクトの使用例
- 課題4-1

# 関数とは

- これまでの演習で、いくつかの関数を使ってきました。
  - print関数、int関数、str関数、input関数、range関数など
- 関数は、プログラム内でよく利用する処理や作業を、手軽に呼び出せるように用意されているもので、Pythonで最初から用意されている関数を**組み込み関数**と呼びます。
- 関数の呼び出し(復習)

- 構文 `戻り値を受け取る変数 = 関数名(引数1, 引数2, ...)`



- 3 -

# 関数を定義する

- Pythonでは新しい関数を自分で作る(定義する)こともできます。
  - 繰り返し利用するような処理や、他のプログラムでも再利用できそうな処理は、関数にまとめておくとプログラムを効率的に作ることができます。
  - 関数定義の構文
    - 関数は**def文**で定義します。
    - defの後に関数名を置き、引数を受け取る変数のリストを丸括弧で囲みます。引数がない場合は丸括弧の中は空にします。
    - 関数内の処理はブロックに書きます。
    - 呼び出し元に結果を返すには**return文**を使います。
  - Spyderのメニューから[ファイル]-[新規ファイル]を選択して、あたらしいソースファイルを作成します。

BMIを計算するcalBMI関数の定義例です。

- 引数として体重、身長(単位:m)を受け取ります。
- 最後にreturn文で変数bmiの値を返します。

```
def calBMI(taiju, shincho):  
    bmi = taiju / shincho / shincho  
    if bmi > 25.0:  
        print("太っている")  
    else:  
        print("太っていない")  
    return bmi
```

- 4 -

# 関数を呼び出す

- いったん定義した関数は、呼び出せば何度でも実行できるので便利です。

- **実引数と仮引数**

- 関数呼び出しで指定する引数のことを**実引数**と呼びます。
- 関数定義の引数のリストに記述した変数を**仮引数**と呼びます。「仮」とは、実際に関数が呼び出されるまでどのような値が入るかわからないからです。

- **ローカル変数**

- 関数の中で使われている変数は関数の中でだけ有効です。
- 関数の中と外で同じ名前の変数があったとしても、別のものとして扱われます。

```
def calBMI(taiju, shincho):  
    bmi = taiju / shincho / shincho  
    if bmi > 25.0:  
        print("太っている")  
    else:  
        print("太っていない")  
    return bmi
```

2つのbmiは別扱い

```
bmi = 0  
a = calBMI(60, 1.6)  
print(a)  
a = calBMI(80, 1.6)  
print(a)  
print(bmi)
```

仮引数

実引数

実引数

```
太っていない  
23.4375  
太っている  
31.25  
0
```

# 引数のキーワード指定とデフォルト値

- 引数のキーワード指定

- 複数の引数をとる関数を呼び出す場合、引数の順序に注意する必要があります。
- 実引数に対して仮引数名をキーワード指定することで、順番にかかわらず目的の仮引数に値を渡すことができます。
- 方法: 引数指定の際に、「仮引数名=実引数」の形式で渡す

```
a = calBMI(shincho=1.6, taiju=90)  
print(a)  
a = calBMI(taiju=40, shincho=1.5)  
print(a)
```

- 引数のデフォルト値

- 関数定義の仮引数の後ろに「=値」として、デフォルト値を設定しておくことができます。デフォルト値が設定された引数は、呼び出す際に省略することができます。

```
def calBMI(taiju, shincho=1.5):  
    bmi = taiju / shincho / shincho  
    ... 以下省略 ...
```

# 複数の戻り値

- 戻り値はreturn文で呼び出し元に返すことができますが、複数の戻り値を戻すこともできます。

- 構文 `return 戻り値1, 戻り値2, ...`

- 呼び出し側での受け取り

- 複数の戻り値を返す関数を呼び出した場合、結果はタプル型で受け取れます。
- タプルでなく単独の変数として受け取るには、=の前に戻り値の数だけ変数をカンマで区切って並べます。

```
def calBMI(taiju, shincho=1.5):  
    bmi = taiju / shincho / shincho  
    if bmi > 25.0:  
        txt = "太っている"  
    else:  
        txt = "太っていない"  
    return bmi, txt  
a = calBMI(60, 1.6)  
print(a)
```

タプルで受け取る場合

(23.4375, '太っていない')

単独の変数で受け取る場合

```
a, b = calBMI(60, 1.6)  
print(a)  
print(b)
```

23.4375  
太っていない

- 7 -

# オブジェクト指向とは

- 現実世界のあらゆるモノは、データと機能を併せ持っている

- 例: エアコン

- データ
  - メーカー名、機種名、何畳用、設定モード、設定温度、...
- 機能
  - 電源のON/OFF、設定モードを変える、設定温度を上げる下げる、...



- オブジェクト指向とは、現実世界の「モノ(オブジェクト)」の利便性を、プログラミング開発に取り入れた考え方



面倒なことはオブジェクト自身に任せて楽をしようという発想

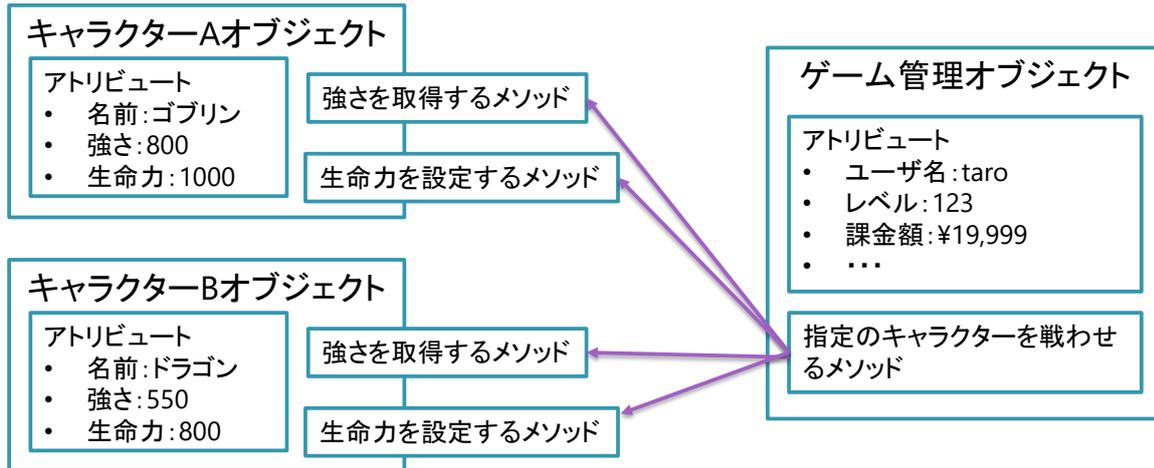
- エアコンを使用している人は、

- 現在のエアコンの設定モードや設定温度を記憶しておく必要はない→設定はエアコン自身が記憶している
- エアコンの中身(機械、しくみ)を知らなくても、リモコンを操作すれば簡単にON/OFF、温度の上下ができる→難しいことはエアコン自身が自分で動いてくれる

- 8 -

# Pythonにおけるオブジェクト(1)

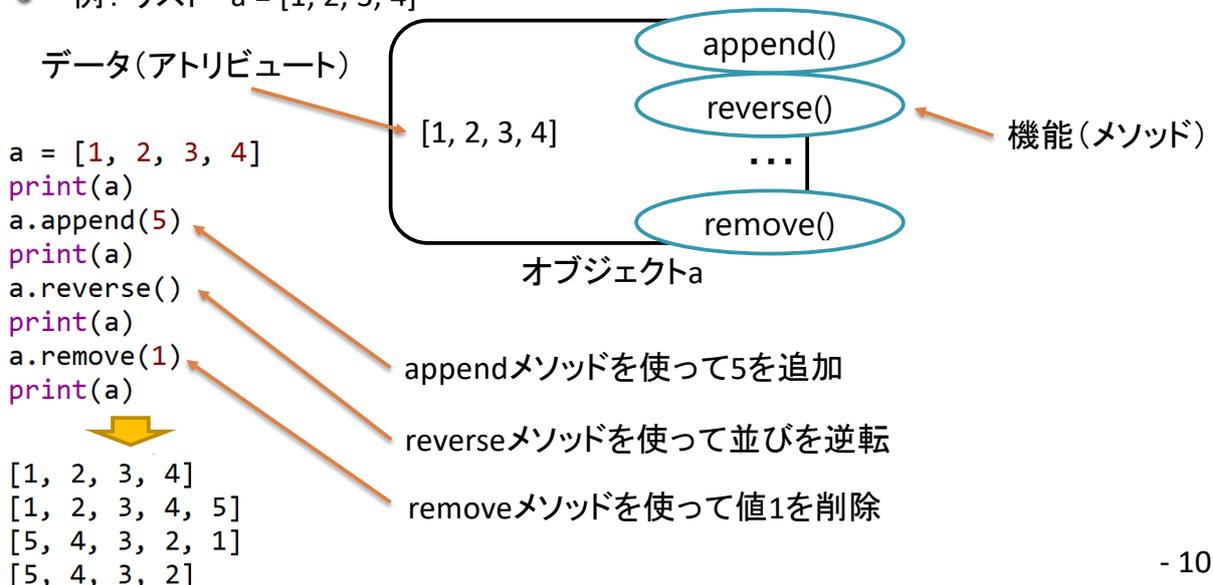
- Pythonなどのオブジェクト指向型のプログラミング言語では、「オブジェクト(Object)」と呼ばれる機能部品を組み合わせてプログラムを開発できます
- オブジェクトには、オブジェクトの特性や状態を記憶しておくためのデータである複数の「アトリビュート(Attribute)」と、アトリビュートに対する処理を実行する複数の「メソッド(Method)」が含まれています
- プログラムを実行すると、オブジェクト同士はお互いに相手のメソッドを呼び出しながら動作します
- メソッドは、オブジェクト専用の関数といえます



- 9 -

# Pythonにおけるオブジェクト(2)

- Pythonでは、すべての組み込みデータ型(数値型、文字列型、リスト等)の変数は、オブジェクトの特徴を持っています。
  - オブジェクトは、データ(アトリビュート)と、データに対する機能(メソッド:オブジェクト専用の関数)を併せ持った存在です。
  - オブジェクトのメソッドを呼び出すには、オブジェクト変数名の後ろに.(ドット)で区切ってメソッド名を指定します。
  - 例: リスト `a = [1, 2, 3, 4]`



- 10 -

# クラス

- Pythonでは組み込み型以外の様々なオブジェクトが利用できます。
  - クラスはオブジェクトの設計図であり、組み込み型以外のオブジェクトは、クラスをもとに生成します。
  - クラスをもとにしたオブジェクトを生成する構文

```
インスタンス変数名 = クラス名(引数)
```

- 演習: 日付を管理するためのdateクラス
  - dateクラスは、Pythonの標準モジュールであるdatetimeモジュールに備わっていますが、そのままでは使用できないので、事前にimportする(読み込む)必要があります。
  - Spyderで[ファイル]-[新規ファイル]として、下記のコードを試しましょう。

```
from datetime import date ← この文法については後ほど解説します
```

```
a = date(2025, 11, 15)
print (a) → 2025-11-15
```

- インスタンスとは？
  - クラスをもとに生成した個々のオブジェクトを、クラスを実体化した存在という意味を明確化するためにインスタンス(実体)と呼ぶ場合があります。

- 11 -

# dateオブジェクトの使用例

- dateクラスのオブジェクトには、便利なメソッドや演算が備わっています。
  - 曜日を数字で取得するweekdayメソッド
    - 月曜日を0、日曜日を6として、曜日を整数で取得できます。
  - 日数計算が可能
    - インスタンスの差(-)で日数が計算可能です。

```
from datetime import date
```

```
a = date(2025, 11, 15)
print (a)
```

```
b = date(2025, 12, 25)
print (b)
```

```
print(b.weekday())
print(b - a)
```

```
曜日の取得 → 2025-11-15
2025-12-25
日数計算 → 3
40 days, 0:00:00
```

- 12 -

# 公式リファレンス

- Python標準ライブラリに用意されている関数やクラスの使い方マニュアル

<https://docs.python.org/ja/3/library/>

docs.python.org/ja/3/library/datetime.html#date-objects

- Naive インスタンス
- 定数
  - MINYEAR
  - MAXYEAR
  - UTC
- 利用可能なデータ型
  - 共通の特徴
  - オブジェクトが Aware なのか Naive なのかの判断
- timedelta オブジェクト
  - timedelta
  - min
  - max
  - resolution
  - days
  - seconds
  - microseconds
  - total\_seconds()
  - 使用例: timedelta

## date オブジェクト

`date` オブジェクトは、両方向に無期限に拡張された現在のグレゴリオ暦という理想化さオを表します。

1年1月1日は日番号 1、1年1月2日は日番号 2 と呼ばれ、他も同様です。 [2]

```
class datetime.date(year, month, day)
```

全ての引数が必須です。引数は整数で、次の範囲に収まっていなければなりません:

- MINYEAR <= year <= MAXYEAR
- 1 <= month <= 12
- 1 <= day <= 指定された月と年における日数

範囲を超えた引数を与えた場合、`ValueError` が送出されます。

- 13 -

## 課題4-1

- p7のcalBMI関数では、身長はm(メートル)で指定する仕様になっていました。
- 私たちの普段の生活では、身長はcmで扱ったほうがわかりやすいので、shinchoをcmで指定できるようにcalBMI関数を修正してみてください。

```
a, b = calBMI(60, 1.6)
```



身長をmでなく  
cmで指定できる  
ように!

```
a, b = calBMI(60, 160)
```

- 14 -