



# Pythonの道

トレーニングドリル⑤

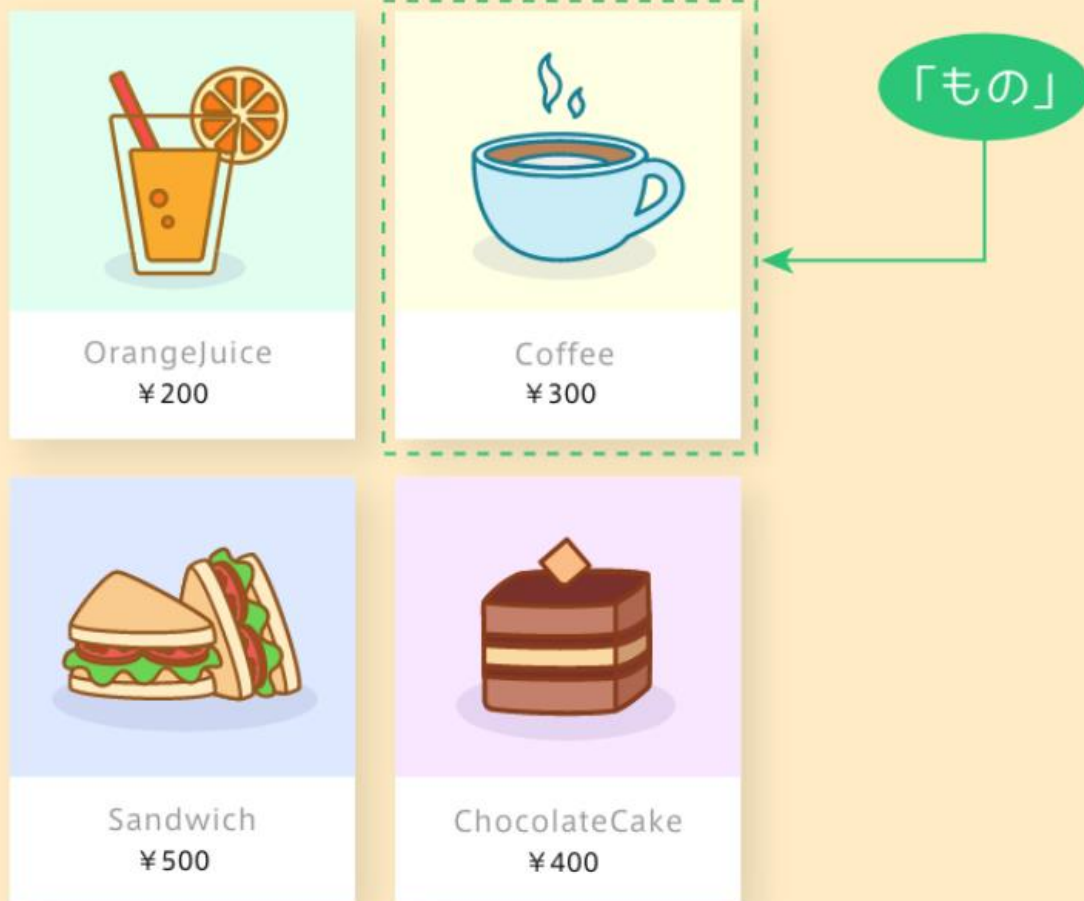
# もくじ

- ・料理注文システムを作ろう



# プログラミングと「もの」

メニューを1つの「もの」として扱うことで、複数のメニューを管理する方法を学んでいく。メニューを1つのかたまりとして見ることを通してクラスという概念を理解しよう。



# 目標物の確認

まず、これから作成する「料理注文システム」を確認してみよう。

「料理注文システム」では、実行中にユーザーから購入する商品の番号と個数を受け取る。

>\_ コンソール

0. サンドイッチ: ¥500  
1. チョコケーキ: ¥400  
2. コーヒー: ¥300  
3. オレンジジュース: ¥200

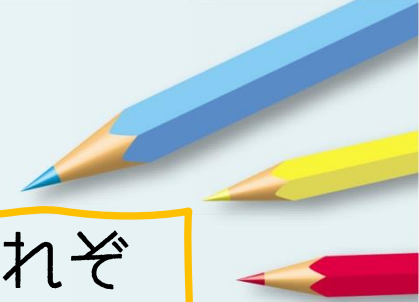


>\_ コンソール

メニューの番号を選択してください: 1  
選択されたメニュー: チョコケーキ  
個数を入力してください (3 つ以上で 1 割引) : 4  
合計は 1440 円です ■

# メニューに注目しよう

先程、確認したシステムのメニューは、下の図のようにそれぞれ「名前」と「値段」という情報をもっている。  
これはメニューという「もの」ごとに情報を持っていることを意味している。



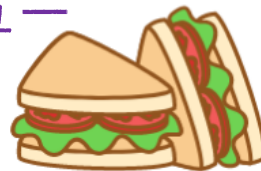
メニュー



情報

- 名前：チョコケーキ
- 値段：¥400

メニュー



情報

- 名前：サンドイッチ
- 値段：¥500

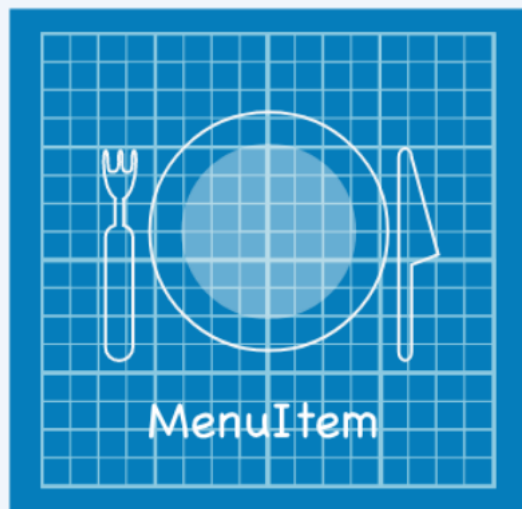
それぞれのメニューは  
「名前」と「値段」という情報を持っている

# クラス：「もの」を生成する仕組み

プログラミングでメニューという「もの」を生成するには、まず、その「設計図」を用意する必要がある。

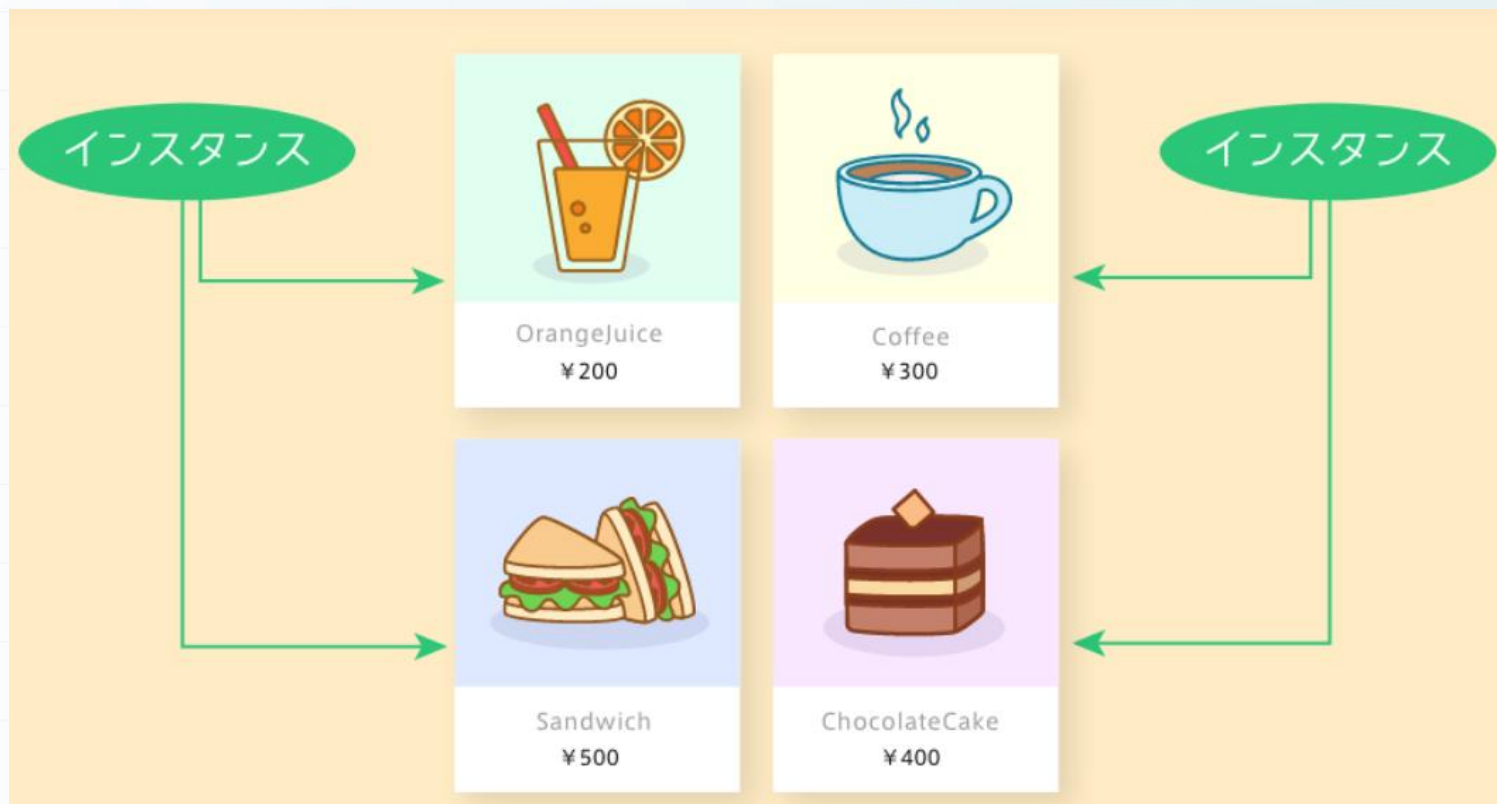
設計図のことをクラス、「もの」のことをインスタンスと呼ぶことを知っておこう。

## クラスのイメージ



設計図  
(クラス)

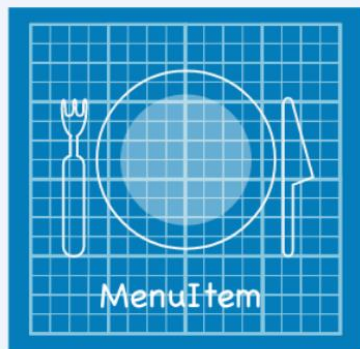
## インスタンスのイメージ



# インスタンスを生成するステップ

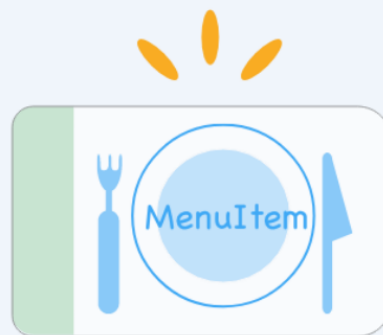
インスタンスは、以下の流れに沿って生成していく。  
まずは、step1の「クラスを用意する」から順番に進めていこう。

## step 1 クラスを用意する



設計図  
(クラス)

## step 2 クラスから インスタンスを生成する



「もの」(空のインスタンス)

## step 3 インスタンスに 情報を追加する



名前: サンドイッチ  
値段: ¥500

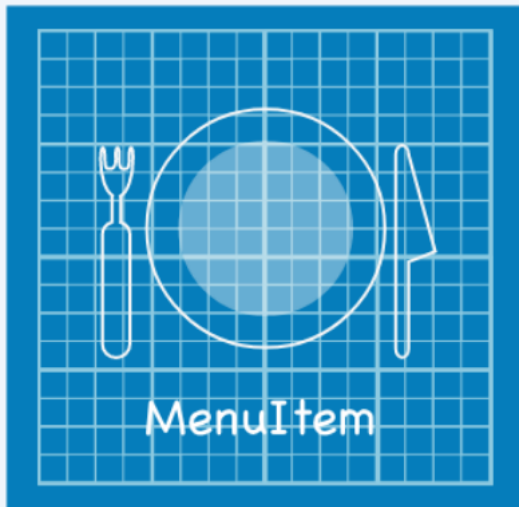
### 情報

- 名前: サンドイッチ
- 値段: ¥500

# クラスの定義

最初にクラス（設計図）を用意する。クラスは「class クラス名:」とすることで定義できる。  
また、クラス名は「MenuItem」のように大文字で始めるようにしよう。

## クラスのイメージ



設計図  
(クラス)

## クラスの定義

class MenuItem: 行末にコロンをつける！

↑  
クラス名は大文字で始める

# クラスの中身

クラスの中身（設計図の内容）は「class MenuItem:」より後の行で、インデントをして書いていく。  
今回はまだ処理を追加する必要がないので「pass」と書く。  
（これは何も処理がないことを表している）。

```
class MenuItem:  
    # 処理
```



```
class MenuItem:  
    pass
```



何も処理がないことを表す

# 練習



## 【問題】

# MenuItemクラスを定義してください



# 練習



【答え】

# MenuItemクラスを定義してください

```
class MenuItem:  
    pass
```

# クラスからインスタンスを生成する

クラス（設計図）が用意できたので、次はMenuItemクラスから、実際のメニュー（インスタンス）を生成する方法を学んでいく。

## インスタンス生成手順

1. クラスを用意する 済
2. クラスからインスタンスを生成する
3. インスタンスに情報を追加する

# インスタンスの生成

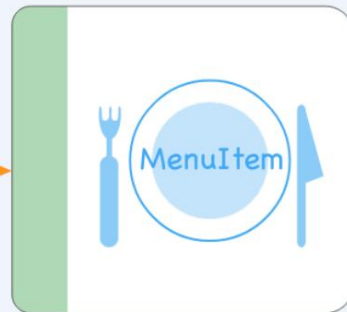
「クラス名()」とそのクラスを呼び出すことで、クラス（設計図）を用いて新しくインスタンスを生成することができる。  
また、「変数名 = クラス名()」とすることで、生成したインスタンスを変数に代入することが可能になる。

インスタンスの生成のイメージ図



MenuItemクラス  
(設計図)

生成！



空のインスタンス  
(もの)

```
class MenuItem:
```

```
    pass
```

```
menu_item1 = MenuItem()
```

変数menu\_item1に代入

MenuItemクラスからインスタンスを生成

# 練習



## 【問題】

```
class MenuItem:  
    pass
```

# MenuItem クラスのインスタンスを生成してください



# 練習



【答え】

```
class MenuItem:  
    pass
```

```
# MenuItem クラスのインスタンスを生成してください  
menu_item1 = MenuItem()
```

# インスタンスに情報を追加しよう

MenuItemクラスから生成したインスタンスに情報を追加する。  
先程、作成したmenu\_item1のインスタンスに「名前がサンドイッチ」という情報を追加する。

## インスタンス生成手順

1. クラスを用意する

済

2. クラスからインスタンスを生成する

済

3. インスタンスに情報を追加する

# インスタンスと情報

それぞれのインスタンス（メニュー）には、様々な情報を追加することができる。図の例では、name(名前)が「サンドイッチ」、price(値段)が「500」という情報を持っている。

## インスタンスと情報



MenuItemクラスの  
インスタンス

### 情報

- name : サンドイッチ
- price : ￥500

# インスタンス変数

「menu\_item1.name="サンドイッチ"」とすることで、menu\_item1に「name」が「サンドイッチ」であるという情報を追加することができる。  
この時、「name」のことを「インスタンス変数」と呼ぶことを知っておこう。

```
class MenuItem:
```

```
    pass
```

← 値を代入

```
menu_item1 = MenuItem()
```

```
menu_item1.name = "サンドイッチ"
```

インスタンス変数

# インスタンス変数を用いる

「インスタンス.インスタンス変数名」とすることで、そのインスタンス変数の値を使うことができる。

```
class MenuItem:  
    pass
```

```
menu_item1 = MenuItem()  
menu_item1.name = " サンドイッチ"  
print(menu_item1.name)
```

インスタンス変数の値を用いる

⇒ サンドイッチ

# 練習



## 【問題】

```
class MenuItem:  
    pass
```

```
menu_item1 = MenuItem()
```

```
menu_item1.name = 'サンドイッチ'  
print(menu_item1.name)
```

# menu\_item1のpriceに「500」を代入してください

# menu\_item1のpriceを出力してください

# 練習



【答え】

```
class MenuItem:  
    pass
```

```
menu_item1 = MenuItem()
```

```
menu_item1.name = 'サンドイッチ'  
print(menu_item1.name)
```

```
# menu_item1のpriceに「500」を代入してください  
menu_item1.price = 500
```

```
# menu_item1のpriceを出力してください  
print(menu_item1.price)
```

# インスタンスごとに情報を持たせる

複数のインスタンスを作り、インスタンスごとに異なる情報が持てることを確認する。



MenuItemクラス  
(設計図)

生成！



サンドイッチ  
¥500 (税込)

生成！



チョコケーキ  
¥400 (税込)

生成！



コーヒー  
¥300 (税込)

生成！



オレンジジュース  
¥200 (税込)

# 練習



## 【問題】

```
class MenuItem:  
    pass
```

```
menu_item1 = MenuItem()
```

```
menu_item1.name = 'サンドイッチ'  
print(menu_item1.name)
```

```
menu_item1.price = 500  
print(menu_item1.price)
```

```
# MenuItemクラスのインスタンスを生成してください
```



次のページに続く

# 練習

# menu\_item2のnameに「チョコケーキ」を代入してください

# menu\_item2の名を出力してください

# menu\_item2のpriceに「400」を代入してください

# menu\_item2のpriceを出力してください

# 練習



【答え】

```
class MenuItem:  
    pass
```

```
menu_item1 = MenuItem()
```

```
menu_item1.name = 'サンドイッチ'  
print(menu_item1.name)
```

```
menu_item1.price = 500  
print(menu_item1.price)
```

# MenuItemクラスのインスタンスを生成してください

```
menu_item2 = MenuItem()
```

# menu\_item2のnameに「チョコケーキ」を代入してください

```
menu_item2.name = 'チョコケーキ'
```

次のページに続く

# 練習

# menu\_item2の名を出力してください  
`print(menu_item2.name)`

# menu\_item2のpriceに「400」を代入してください  
`menu_item2.price = 400`

# menu\_item2のpriceを出力してください  
`print(menu_item2.price)`

# クラスの中に処理を追加する

クラスの中では関数を定義することができる。クラスの中で定義した関数のことをメソッドという。

メソッドの定義方法は通常関数と同じであるが、第1引数にselfを追加する必要があることに注意する。

```
class MenuItem:
```

```
    def hello(self):  
        print("こんにちは")
```

必ず第1引数にselfを追加する

クラスの中でメソッド（関数）を定義

# クラスの中で定義したメソッドを呼び出す

クラスの中で定義したメソッドは、インスタンスに対して使うように呼び出す。「インスタンス.メソッド名()」とすることで、そのメソッドを呼び出すことができる。クラスの中で定義し、インスタンスの中で呼び出すメソッドのことを**インスタンスメソッド**という。

```
class MenuItem:  
    def hello(self):  
        print("こんにちは")  
menu_item1 = MenuItem()  
menu_item1.hello()
```

クラスの中で定義したメソッドを呼び出す

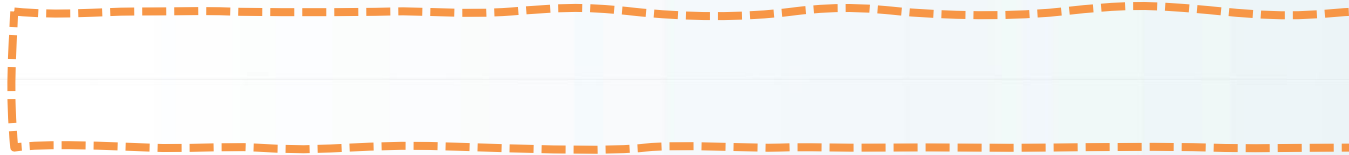
⇒ こんにちは

# 練習



## 【問題】

```
class MenuItem:  
    # infoメソッドを定義してください
```



```
menu_item1 = MenuItem()  
menu_item1.name = 'サンドイッチ'  
menu_item1.price = 500
```

```
# menu_item1に対してinfoメソッドを呼び出してください
```



```
menu_item2 = MenuItem()  
menu_item2.name = 'チョコケーキ'  
menu_item2.price = 400
```

```
# menu_item2に対してinfoメソッドを呼び出してください
```



# 練習



【答え】

```
class MenuItem:
    # infoメソッドを定義してください
    def info(self):
        print('メニューの名前と値段が表示されます')

menu_item1 = MenuItem()
menu_item1.name = 'サンドイッチ'
menu_item1.price = 500

# menu_item1に対してinfoメソッドを呼び出してください
menu_item1.info()

menu_item2 = MenuItem()
menu_item2.name = 'チョコケーキ'
menu_item2.price = 400

# menu_item2に対してinfoメソッドを呼び出してください
menu_item2.info()
```

# メニューの内容を出力する機能

先程、定義した「infoメソッド」を改良して、そのメニューの名前や値段を出力するようにしてみよう。そのために、呼び出しているインスタンスのインスタンス変数を使う必要がある。

```
class MenuItem:
```

```
    def info(self): 呼び出したインスタンスが代入されている  
        print(" メニューの情報が表示されます。")
```

```
    .  
    .  
    .
```

```
menu_item1.info()
```

# インスタンスメソッドのselfの正体

インスタンスメソッドの第1引数に指定した「self」には、そのメソッドを呼び出したインスタンス自身が代入されている。そのため、図のようにメソッド内で「self.name」とすることで、そのメソッドを呼び出している「menu\_item1」の「name」の値を取得することができる。

```
class MenuItem:
```

```
    def info(self): menu_item1が代入される  
        print(self.name)
```

```
menu_item1 = MenuItem()
```

```
menu_item1.name = " サンドイッチ"
```

```
menu_item1.info()
```

⇒ サンドイッチ menu\_item1.nameの値

# 型変換の復習

今回は「name」と「price」の値を用いて、「サンドイッチ：¥500」と出力してみよう。

この時、「price」の値は数値型であるため、文字列と結合するためには文字列型に変換する必要がある。まずは、数値を文字列に変換する方法を再確認しよう。

```
age = 21
```

```
print(str(age) + " 歳です" )
```

文字列に変換

⇒ 21歳です

# 練習



## 【問題】

```
class MenuItem:  
    def info(self):  
        # 「〇〇: ¥□□」となるように出力してください
```

```
menu_item1 = MenuItem()  
menu_item1.name = 'サンドイッチ'  
menu_item1.price = 500
```

```
menu_item1.info()
```

```
menu_item2 = MenuItem()  
menu_item2.name = 'チョコケーキ'  
menu_item2.price = 400
```

```
menu_item2.info()
```

# 練習



【答え】

```
class MenuItem:  
    def info(self):  
        # 「〇〇: ¥□□」となるように出力してください  
        print(self.name + ': ¥' + str(self.price))  
        全角で記載する！
```

```
menu_item1 = MenuItem()  
menu_item1.name = 'サンドイッチ'  
menu_item1.price = 500
```

```
menu_item1.info()
```

```
menu_item2 = MenuItem()  
menu_item2.name = 'チョコケーキ'  
menu_item2.price = 400
```

```
menu_item2.info()
```

# インスタンスメソッドの活用

インスタンスメソッドは通常の変数と同様に、戻り値を返すこともできる。「return」を使って、infoメソッドが戻り値を返せるようにしてみよう。

```
class MenuItem:
    def info(self):
        # print()の中身を戻り値として返してください
        return self.name + ': ¥' + str(self.price)
```

```
menu_item1 = MenuItem()
menu_item1.name = 'サンドイッチ'
menu_item1.price = 500
```

```
# menu_item1.info()の値を出力してください
print(menu_item1.info())
```

```
menu_item2 = MenuItem()
menu_item2.name = 'チョコケーキ'
menu_item2.price = 400
```

```
# menu_item2.info()の値を出力してください
print(menu_item2.info())
```

```
サンドイッチ: ¥500
チョコケーキ: ¥400
>>>
```

# 練習



## 【問題】

```
class MenuItem:  
    def info(self):  
        # print()の中身に戻り値として返してください
```

```
menu_item1 = MenuItem()  
menu_item1.name = 'サンドイッチ'  
menu_item1.price = 500
```

```
# menu_item1.info()の値を出力してください
```

```
menu_item2 = MenuItem()  
menu_item2.name = 'チョコケーキ'  
menu_item2.price = 400
```

```
# menu_item2.info()の値を出力してください
```

# 練習



【答え】

```
class MenuItem:
    def info(self):
        # print()の中身に戻り値として返してください
        return self.name + ':' + '¥' + str(self.price)
```

全角で記載する！

```
menu_item1 = MenuItem()
menu_item1.name = 'サンドイッチ'
menu_item1.price = 500
```

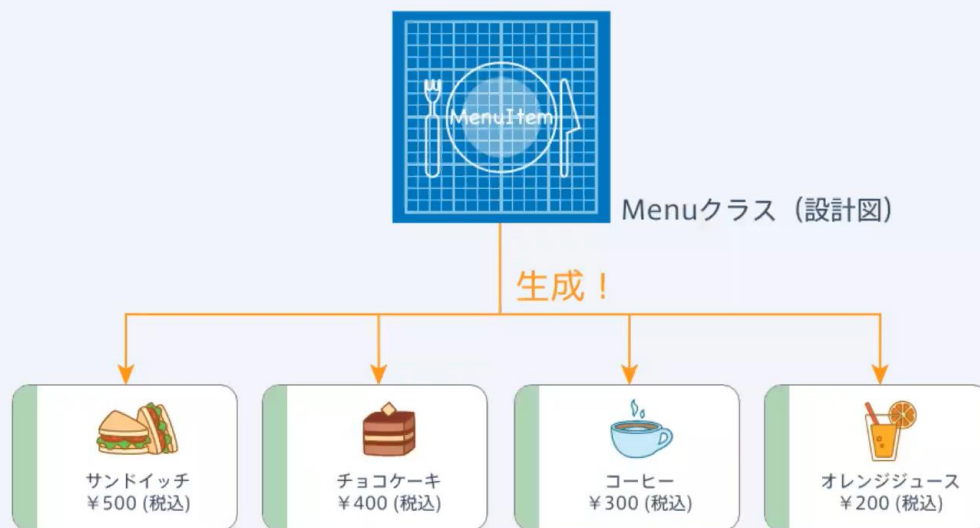
```
# menu_item1.info()の値を出力してください
print(menu_item1.info())
```

```
menu_item2 = MenuItem()
menu_item2.name = 'チョコケーキ'
menu_item2.price = 400
```

```
# menu_item2.info()の値を出力してください
print(menu_item2.info())
```

# クラスとインスタンスのおさらい

ここで一度、クラスとインスタンスについておさらいする。  
クラスは設計図のようなもので、その設計図から作る実際の「もの」にあたるものがインスタンスである。  
インスタンスは情報として「インスタンス変数」を処理として「インスタンスメソッド」を持っている。





サンドイッチ  
¥500 (税込)

情報

メソッド

- 名前：サンドイッチ
- 値段：¥500

- infoメソッド

# 合計金額を求めるメソッド

ここまで学んだことを活かして、メニューの合計金額を求めるメソッドを作ってみよう。インスタンスメソッドに引き数を渡す場合、メソッドの定義側では「self」の分だけ引数の順番がずれることに注意する。

```
class MenuItem:
```

```
    .
```

```
    .
```

```
    def get_total_price(self, count):
```

```
    .
```

```
    .
```

```
menu_item1.get_total_price(4)
```

「4」が代入される

# 練習



## 【問題】

```
class MenuItem:  
    def info(self):  
        return self.name + ': ¥' + str(self.price)
```

# get\_total\_priceメソッドを定義してください

```
menu_item1 = MenuItem()  
menu_item1.name = 'サンドイッチ'  
menu_item1.price = 500
```

```
print(menu_item1.info())
```

# get\_total\_priceメソッドを呼び出してください

# 「合計は〇〇円です」となるように出力してください

# 練習



【答え】

```
class MenuItem:
    def info(self):
        return self.name + ': ¥' + str(self.price)

    # get_total_priceメソッドを定義してください
    def get_total_price(self, count):
        total_price = self.price * count
        return total_price

menu_item1 = MenuItem()
menu_item1.name = 'サンドイッチ'
menu_item1.price = 500

print(menu_item1.info())

# get_total_priceメソッドを呼び出してください
result = menu_item1.get_total_price(4)

# 「合計は〇〇円です」となるように出力してください
print('合計は' + str(result) + '円です')
```

# 特殊なインスタンスメソッド

インスタンスを生成した直後に処理を実行することができる、  
「\_\_init\_\_メソッド」は、「クラス名()」でインスタンスを生成した直後に自動で呼び出される。

\_\_init\_\_メソッド

「MenuItem()」が実行される

def get\_total\_price(self, count):

\_\_init\_\_メソッドがあれば

生成されたインスタンスに対して  
\_\_init\_\_メソッドが自動で呼び出される！

# \_\_init\_\_メソッドの使い方

\_\_init\_\_メソッドは、他のインスタンスメソッドと同じように定義することができる。「menu\_item1 = MenuItem()」でMenuItemクラスのインスタンスが生成された直後に\_\_init\_\_メソッドが呼び出され、その中の処理が実行される。

```
class MenuItem:
```

```
    def __init__(self):  
        print("メニューが生成されました")
```

```
    def get_total_price(self, count):
```

・ インスタンスが生成されると自動で呼び出される  
・

```
menu_item1 = MenuItem()
```

⇒ メニューが生成されました

# 練習

## 【問題】

```
class MenuItem:
```

```
    # __init__メソッドを定義してください
```

```
    def info(self):
```

```
        return self.name + ': ¥' + str(self.price)
```

```
    def get_total_price(self, count):
```

```
        total_price = self.price * count
```

```
        return total_price
```

```
menu_item1 = MenuItem()
```

```
menu_item1.name = 'サンドイッチ'
```

```
menu_item1.price = 500
```

```
print(menu_item1.info())
```

```
result = menu_item1.get_total_price(4)
```

```
print('合計は' + str(result) + '円です')
```

# 練習



【答え】

```
class MenuItem:
```

```
    # __init__メソッドを定義してください
```

```
    def __init__(self):
```

```
        print('MenuItemクラスのインスタンスが生成されました！')
```

```
    def info(self):
```

```
        return self.name + ': ¥' + str(self.price)
```

```
    def get_total_price(self, count):
```

```
        total_price = self.price * count
```

```
        return total_price
```

```
menu_item1 = MenuItem()
```

```
menu_item1.name = 'サンドイッチ'
```

```
menu_item1.price = 500
```

```
print(menu_item1.info())
```

```
result = menu_item1.get_total_price(4)
```

```
print('合計は' + str(result) + '円です')
```

# インスタンス変数に値を代入する

`__init__` メソッドはどういう時に使えばいいのだろうか？  
このメソッドを使うことでインスタンス生成すると同時にインスタンス変数に値を代入することができる。図の2と3の処理を同時に行うことができる。

## インスタンス生成手順

1. クラスを用意する
2. クラスからインスタンスを生成する
3. インスタンスに情報を追加する

2と3をまとめて、「インスタンスを生成すると同時にインスタンス変数に値を代入する」

# インスタンス変数に値を代入する

`__init__`メソッドの中で、インスタンス変数に値を代入してみよう。インスタンスメソッドの中では、「`self.変数名`」でインスタンス変数を扱うことができる。「`self.変数名=値`」でインスタンス変数に値を代入してみよう。

```
class MenuItem:
```

```
    def __init__(self):  
        print("メニューが生成されました")
```

```
    self.name = " サンドイッチ"
```

```
menu_item1 = MenuItem()  
print(menu_item1.name)
```

インスタンス、生成直後に" サンドイッチ" をインスタンス変数`name`に代入

⇒ サンドイッチ

# 練習

## 【問題】

```
class MenuItem:
    def __init__(self):
        # self.nameに「サンドイッチ」を代入してください
        # self.priceに「500」を代入してください
    def info(self):
        return self.name + ': ¥' + str(self.price)

    def get_total_price(self, count):
        total_price = self.price * count
        return total_price

menu_item1 = MenuItem()

print(menu_item1.info())

result = menu_item1.get_total_price(4)
print('合計は' + str(result) + '円です')
```

# 練習



【答え】

```
class MenuItem:
    def __init__(self):
        # self.nameに「サンドイッチ」を代入してください
        self.name = 'サンドイッチ'

        # self.priceに「500」を代入してください
        self.price = 500

    def info(self):
        return self.name + ': ¥' + str(self.price)

    def get_total_price(self, count):
        total_price = self.price * count
        return total_price

menu_item1 = MenuItem()
print(menu_item1.info())

result = menu_item1.get_total_price(4)
print('合計は' + str(result) + '円です')
```

# \_\_init\_\_メソッドに引数を渡そう

先程の方法だと、どのインスタンスもnameが「サンドイッチ」になってしまう。\_\_init\_\_メソッドに引数を渡すことで、インスタンスごとに値を変えることができる。その際、インスタンスを生成している「クラス名()」に対して引数を渡すことで、\_\_init\_\_メソッドにその値を渡すことができる。

```
class MenuItem:
```

```
    def __init__(self, name): チョコレート ←  
        self.name = name
```

```
menu_item1 = MenuItem(" チョコレート ")  
print(menu_item1.name)
```

⇒ チョコレート

# 練習



## 【問題】

```
class MenuItem:
```

```
    # 引数「name」と「price」を受け取るようにしてください
```

```
        # 「サンドイッチ」の代わりに引数nameの値を代入してください
```

```
        # 「500」の代わりに引数priceの値を代入してください
```

```
    def info(self):
```

```
        return self.name + ': ¥' + str(self.price)
```

```
    def get_total_price(self, count):
```

```
        total_price = self.price * count
```

```
        return total_price
```

```
# 引数を「サンドイッチ」と「500」としてください
```

```
print(menu_item1.info())
```

```
result = menu_item1.get_total_price(4)
```

```
print('合計は' + str(result) + '円です')
```

# 練習



【答え】

```
class MenuItem:
```

```
    # 引数「name」と「price」を受け取るようにしてください
```

```
    def __init__(self, name, price):
```

```
        # 「サンドイッチ」の代わりに引数nameの値を代入してください
```

```
        self.name = name
```

```
        # 「500」の代わりに引数priceの値を代入してください
```

```
        self.price = price
```

```
    def info(self):
```

```
        return self.name + ': ¥' + str(self.price)
```

```
    def get_total_price(self, count):
```

```
        total_price = self.price * count
```

```
        return total_price
```

```
# 引数を「サンドイッチ」と「500」としてください
```

```
menu_item1 = MenuItem('サンドイッチ', 500)
```

```
print(menu_item1.info())
```

```
result = menu_item1.get_total_price(4)
```

```
print('合計は' + str(result) + '円です')
```

# オブジェクトを分割する

コードを分割して管理しやすくする。まず、MenuItemクラスの定義部分を「menu\_item.py」という別のファイルに移してみよう。

script.py



プログラムの実行部分

menu\_item.py



MenuItemクラスの定義部分

# importの復習

Pythonではコードの一部を別ファイルに移して、モジュールとして読み込むことができる。モジュールを読み込むには「import モジュール名（ファイル名）」とする。

script.py

```
import menu_item 「menu_item.py」の読み込み ←  
menu_item1 = menu_item.Menuitem(...)  
               「menu_item.py」の中のMenuitemクラス
```

menu\_item.py

```
class Menuitem:  
    def __init__(self,name,price):  
        self.name = name  
        self.price = price
```

# クラスを直接、読み込む

importには、より便利な方法が用意されている。「from モジュール名 import クラス名」とすることで、そのモジュール内の指定したクラスを直接読み込むことができる。このようにすることで、「script.py」内でも今までと同じようにMenuItemクラスを用いることができる。

script.py

```
from menu_item import MenuItem
```

モジュール名                      クラス名

```
menu_item1 = MenuItem(" サンドイッチ" ,500)
```

そのままMenuItemクラスが使える!

menu\_item.py

```
class MenuItem:
```

```
    def __init__(self,name,price):  
        self.name = name  
        self.price = price
```

# 練習



【問題】 menu\_item.py

# MenuItemクラスのコードを貼り付けてください

【問題】 script.py

# menu\_item.pyからMenuItemクラスを読み込んでください

```
menu_item1 = MenuItem('サンドイッチ', 500)
```

```
print(menu_item1.info())
```

```
result = menu_item1.get_total_price(4)  
print('合計は' + str(result) + '円です')
```

# 練習



【答え】 menu\_item.py

# MenuItemクラスのコードを貼り付けてください

class MenuItem:

def \_\_init\_\_(self, name, price):

self.name = name

self.price = price

def info(self):

return self.name + ': ¥' + str(self.price)

def get\_total\_price(self, count):

total\_price = self.price \* count

return total\_price

【答え】 script.py

# menu\_item.pyからMenuItemクラスを読み込んでください

from menu\_item import MenuItem

menu\_item1 = MenuItem('サンドイッチ', 500)

print(menu\_item1.info())

result = menu\_item1.get\_total\_price(4)

print('合計は' + str(result) + '円です')

# メニュー一覧を表示しよう

MenuItemクラスの定義部分を「menu\_item.py」に分けることができたので、「料理注文システム」を作っていく。まず、それぞれのメニューを一覧で表示できるようにする。複数メニューのインスタンスをリストに入れて、そのリストに対して繰り返す処理を作る。

## メニュー一覧

サンドイッチ:¥500  
チョコケーキ:¥400  
コーヒー:¥300  
オレンジジュース:¥500

# インスタンスのリスト

MenuItemクラスから生成したインスタンスも、リストの要素にすることが可能。まず各インスタンスを要素とするリストを変数menu\_itemsに代入する。そのリストに対してfor文を用いることで、それぞれのインスタンスの情報を出力することができる。

```
from menu_item import MenuItem
```

```
menu_item1 = MenuItem(" サンドイッチ" ,500)
```

```
menu_item2 = MenuItem(" チョコケーキ" ,400)
```

```
menu_items = [menu_item1, menu_item2, ...]
```

```
for menu_item in menu_items:  
    print(menu_item.info())
```

for文を用いて1つずつ  
メニューを出力

⇒ サンドイッチ：¥500  
チョコケーキ：¥400  
.  
.

# 練習



【問題】 menu\_item.py

```
class MenuItem:
```

```
    def __init__(self, name, price):
```

```
        self.name = name
```

```
        self.price = price
```

```
    def info(self):
```

```
        return self.name + ': ¥' + str(self.price)
```

```
    def get_total_price(self, count):
```

```
        total_price = self.price * count
```

```
        return total_price
```

【問題】 script.py

```
from menu_item import MenuItem
```

```
menu_item1 = MenuItem('サンドイッチ', 500)
```

```
menu_item2 = MenuItem('チョコケーキ', 400)
```

```
menu_item3 = MenuItem('コーヒー', 300)
```

```
menu_item4 = MenuItem('オレンジジュース', 200)
```

次のページに続く

# 練習

# 指定されたリストを変数menu\_itemsに代入してください

# for文を作成してください

# 練習



```
【答え】 menu_item.py
class MenuItem:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def info(self):
        return self.name + ': ¥' + str(self.price)

    def get_total_price(self, count):
        total_price = self.price * count
        return total_price
```

```
【答え】 script.py
from menu_item import MenuItem

menu_item1 = MenuItem('サンドイッチ', 500)
menu_item2 = MenuItem('チョコケーキ', 400)
menu_item3 = MenuItem('コーヒー', 300)
menu_item4 = MenuItem('オレンジジュース', 200)
```

次のページに続く

# 練習

# 指定されたリストを変数menu\_itemsに代入してください

```
menu_items = [menu_item1, menu_item2, menu_item3,  
menu_item4]
```

# for文を作成してください

```
for item in menu_items:  
    print(item.info())
```

# メニューの番号を表示しよう

メニューの情報を1つずつ表示できるようになった。次はそれぞれのメニューに0から始まる番号をつけて表示してみよう。

## メニュー一覧

- 0. サンドイッチ: ¥500
- 1. チョコケーキ: ¥400
- 2. コーヒー: ¥300
- 3. オレンジジュース: ¥500

# 繰り返し処理で番号をつける

番号をつけるために番号を保存するための変数（変数index）をfor文の外で用意する。for文の処理の中で値を1だけ増やして更新していく。

```
lessons = [" Python" , " Ruby" , " Java" ]
```

```
index = 0
```

```
for lesson in lessons:
```

```
    print(str(index) + " . " + lesson)
```

```
    index = index + 1
```

indexの値に1を足す

⇒ 0.Python

1.Ruby

2.Java

変数indexの値

# 練習



【問題】 menu\_item.py

```
class MenuItem:
```

```
    def __init__(self, name, price):
```

```
        self.name = name
```

```
        self.price = price
```

```
    def info(self):
```

```
        return self.name + ': ¥' + str(self.price)
```

```
    def get_total_price(self, count):
```

```
        total_price = self.price * count
```

```
        return total_price
```

【答え】 script.py

```
from menu_item import MenuItem
```

```
menu_item1 = MenuItem('サンドイッチ', 500)
```

```
menu_item2 = MenuItem('チョコケーキ', 400)
```

```
menu_item3 = MenuItem('コーヒー', 300)
```

```
menu_item4 = MenuItem('オレンジジュース', 200)
```

次のページに続く

# 練習

```
menu_items = [menu_item1, menu_item2, menu_item3,  
menu_item4]
```

# 変数indexを定義し、数値の「0」を代入してください

```
for item in menu_items:
```

# 「0. サンドイッチ: ¥500」となるように出力してください

# 変数indexに1を加えてください

# 練習



```
【答え】 menu_item.py
class MenuItem:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def info(self):
        return self.name + ': ¥' + str(self.price)

    def get_total_price(self, count):
        total_price = self.price * count
        return total_price
```

```
【答え】 script.py
from menu_item import MenuItem

menu_item1 = MenuItem('サンドイッチ', 500)
menu_item2 = MenuItem('チョコケーキ', 400)
menu_item3 = MenuItem('コーヒー', 300)
menu_item4 = MenuItem('オレンジジュース', 200)
```

次のページに続く

# 練習

```
menu_items = [menu_item1, menu_item2, menu_item3,  
menu_item4]
```

# 変数indexを定義し、数値の「0」を代入してください

```
index = 0
```

```
for item in menu_items:
```

# 「0. サンドイッチ: ¥500」となるように出力してください

```
print(str(index) + '. ' + item.info())
```

# 変数indexに1を加えてください

```
index = index + 1
```

# 入力を受け取る

「input」を用いて、注文するメニューの番号をコンソールから受け取れるようにする。この際、受け取った数字は文字列型になっているので「int()」を用いて数値型に変換する必要がある。

```
menu_items = [menu_item1, menu_item2, . . .]
```

```
order = int(input(" メニューの番号を入力してください："))
```

数値に変換

コンソールから入力を受け取る

```
selected_menu = menu_items[order]
```

選択されたメニュー

# 練習



【問題】 menu\_item.py

```
class MenuItem:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def info(self):
        return self.name + ': ¥' + str(self.price)

    def get_total_price(self, count):
        total_price = self.price * count
        return total_price
```

【答え】 script.py

```
from menu_item import MenuItem

menu_item1 = MenuItem('サンドイッチ', 500)
menu_item2 = MenuItem('チョコケーキ', 400)
menu_item3 = MenuItem('コーヒー', 300)
menu_item4 = MenuItem('オレンジジュース', 200)
```

次のページに続く

# 練習

```
menu_items = [menu_item1, menu_item2, menu_item3,  
menu_item4]
```

```
index = 0
```

```
for item in menu_items:  
    print(str(index) + '. ' + item.info())  
    index += 1
```

```
print('-----')
```

# コンソールから入力を受け取り、変数orderに代入してください

# 選択されたメニューを変数selected\_menuに代入してください

# 「選択されたメニュー: ○○」と出力してください

# 練習



```
【答え】 menu_item.py
class MenuItem:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def info(self):
        return self.name + ': ¥' + str(self.price)

    def get_total_price(self, count):
        total_price = self.price * count
        return total_price
```

```
【答え】 script.py
from menu_item import MenuItem
menu_item1 = MenuItem('サンドイッチ', 500)
menu_item2 = MenuItem('チョコケーキ', 400)
menu_item3 = MenuItem('コーヒー', 300)
menu_item4 = MenuItem('オレンジジュース', 200)
```

次のページに続く

# 練習

```
menu_items = [menu_item1, menu_item2, menu_item3,  
menu_item4]
```

```
index = 0
```

```
for item in menu_items:  
    print(str(index) + '. ' + item.info())  
    index = index + 1
```

```
print('-----')
```

# コンソールから入力を受け取り、変数orderに代入してください

```
order = int(input('メニューの番号を入力してください: '))
```

# 選択されたメニューを変数selected\_menuに代入してください

```
selected_menu = menu_items[order]
```

# 「選択されたメニュー: ○○」と出力してください

```
print('選択されたメニュー: ' + selected_menu.name)
```

# 合計金額を出力する

次に注文する個数を受け取って合計金額を出力できるようにする。合計金額を出力するために「get\_total\_price」を使う。また、注文数が3つ以上のときは10%割引されるようにしてみよう。

script.py

```
result = selected_menu.get_total_price(4)
print(result)
#出力結果：1440.0
```

menu\_item.py

```
class MenuItem:
    def get_total_price(self, count):
        total_price = self.price * count
        if countが3以上:
            total_price = total_price * 0.9
        return total_price
```

変数total\_priceに0.9をかけて値を更新

# 四捨五入

整数と少数を計算した結果は、小数で出力される。今回は料金なので整数で出力されるようにする。round(小数)とすることで、その小数を四捨五入した結果の整数を取得することができる。

```
number = 7 * 0.9  
print(number)
```

⇒ 6.3

小数で出力される

```
number = 7 * 0.9  
print(round(number))
```

6.3を四捨五入

⇒ 6

整数で出力される

# 練習



【問題】 menu\_item.py

```
class MenuItem:
```

```
    def __init__(self, name, price):
```

```
        self.name = name
```

```
        self.price = price
```

```
    def info(self):
```

```
        return self.name + ': ¥' + str(self.price)
```

```
    def get_total_price(self, count):
```

```
        total_price = self.price * count
```

# countが3以上のとき、total\_priceに0.9をかけてください

# total\_priceを四捨五入して、returnしてください

次のページに続く

# 練習



```
【問題】 script.py
from menu_item import MenuItem

menu_item1 = MenuItem('サンドイッチ', 500)
menu_item2 = MenuItem('チョコケーキ', 400)
menu_item3 = MenuItem('コーヒー', 300)
menu_item4 = MenuItem('オレンジジュース', 200)

menu_items = [menu_item1, menu_item2, menu_item3,
menu_item4]

index = 0

for item in menu_items:
    print(str(index) + '. ' + item.info())
    index += 1

print('-----')

order = int(input('メニューの番号を入力してください: '))
selected_menu = menu_items[order]
```

次のページに続く

# 練習

```
print('選択されたメニュー: ' + selected_menu.name)
```

# コンソールから入力を受け取り、変数countに代入してください

# get\_total\_priceメソッドを呼び出してください

# 「合計は〇〇円です」となるように出力してください

# 練習



```
【答え】 menu_item.py
class MenuItem:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def info(self):
        return self.name + ': ¥' + str(self.price)

    def get_total_price(self, count):
        total_price = self.price * count

# countが3以上のとき、total_priceに0.9をかけてください
    if count >= 3:
        total_price = total_price * 0.9

# total_priceを四捨五入して、returnしてください
    return round(total_price)
```

次のページに続く

# 練習



```
【答え】 script.py
from menu_item import MenuItem

menu_item1 = MenuItem('サンドイッチ', 500)
menu_item2 = MenuItem('チョコケーキ', 400)
menu_item3 = MenuItem('コーヒー', 300)
menu_item4 = MenuItem('オレンジジュース', 200)

menu_items = [menu_item1, menu_item2, menu_item3,
menu_item4]

index = 0

for item in menu_items:
    print(str(index) + '. ' + item.info())
    index += 1

print('-----')

order = int(input('メニューの番号を入力してください: '))
selected_menu = menu_items[order]
```

次のページに続く

# 練習



```
print('選択されたメニュー: ' + selected_menu.name)
# コンソールから入力を受け取り、変数countに代入してください
count = int(input('個数を入力してください(3つ以上で1割引): '))
# get_total_priceメソッドを呼び出してください
result = selected_menu.get_total_price(count)

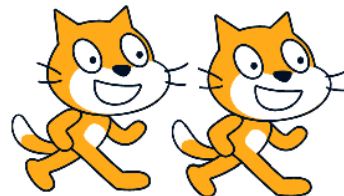
# 「合計は〇〇円です」となるように出力してください
print('合計は' + str(result) + '円です')
```

# 復習 & チャレンジ

ここまで習ったことをScratchでもできるかチャレンジしてみよう。

その過程でScratchでできること、Pythonでないとできないことを整理してみよう。

例えば、データの型という概念はscratchにはあっただろうか？



# メモ



# プログラミング教室の テクノロ

なまえ：