Pythonで始める！
機械学習エンジニア入門

SkillUP AI

配布物

- Day2_lecture.pdf
  - 学習用スライド
- Day2_notebook.ipynb
  - コードが記載されたNotebook
- ReadFirst.txt
  - 確認テストのURLリンクが記載されたテキストファイル
- 改訂履歴.txt
  - 教材の改定履歴

講座の目次

- Day1：実行環境の整備・Python基礎文法①
- Day2：Python基礎文法②
- Day3：データの整理 - NumPy
- Day4：データの整理 - Pandas
- Day5：データの可視化 - Matplotlib
- Day6：データの可視化 - Seaborn
- Day7：データの前処理
- Day8：機械学習モデルの構築と評価

前回の復習

前回の講義で何を学びましたか？

Day2

Python基礎文法②

機械学習に必要なPythonの基礎文法

1. 変数
2. データの型
3. 制御文（if・for・while）
4. 関数
5. クラス

本日はこの２つを理解しましょう！

関数

関数

関数、、、と聞いて何を思い浮かべますか？

関数

関数、、、と聞いて何を思い浮かべますか？

$$y = f(x) = x^2$$

入力に対して処理を施して何かしらを出力する箱！

そもそもなぜこんなものが必要なのか？

関数

まずは関数の必要性を理解しよう！

```
a = (2**2 + 3**2)**(1/2)
b = (4**2 + 5**2)**(1/2)
c = (6**2 + 7**2)**(1/2)
print(a)
print(b)
print(c)
```

```
3.605551275463989
6.4031242374328485
9.219544457292887
```

このコードを見てどう思いますか？

関数

まずは関数の必要性を理解しよう！

```
a = (2**2 + 3**2)**(1/2)
b = (4**2 + 5**2)**(1/2)
c = (6**2 + 7**2)**(1/2)
print(a)
print(b)
print(c)
```

```
3.605551275463989
6.403124237432848 5
9.219544457292887
```

- 同じことを何度も書いてる・・・
- **の所打ち間違えそう
- 一見何をしているコードなのか分からない

この非効率なコードをなんとかしたい！

関数

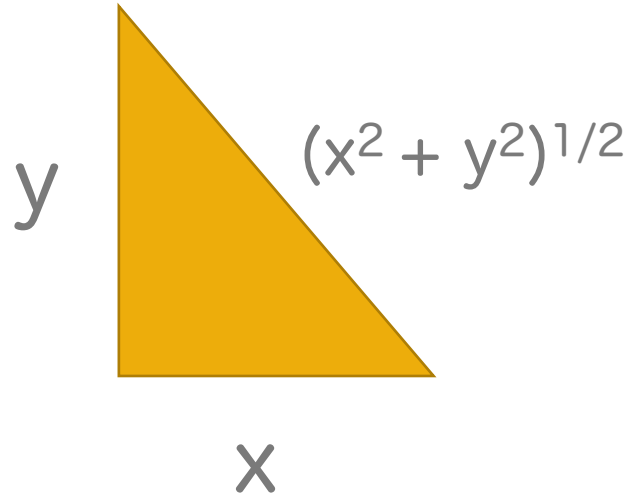
解決策：「入力」と「処理」を切り分ける！

x

y

$(x^2 + y^2)^{1/2}$

参考）



２つ数値を渡してくれれば、処理はこちらがするよ

実は直角三角形の斜辺の長さを求めています

calc_hypotenuse(x, y)というコードで呼び出してね

関数

calc_hypotenuse(x,y)をひとまず実装してみよう！説明は後ほど！

斜辺（hypotenuse）の長さを計算する（calc）する関数

```
def calc_hypotenuse(x,y):
    ans = (x**2 + y**2)**(1/2)
    return ans
```

関数

calc_hypotenuse(x,y)に値を渡せば、中で処理が行われ結果が返ってくる

```
a = calc_hypotenuse(2,3)
b = calc_hypotenuse(4,5)
c = calc_hypotenuse(6,7)
print(a)
print(b)
print(c)
```

```
3.605551275463989
6.403124237432848 5
9.219544457292887
```

機能の実装は１度だけ！後は入力を渡す！

それでは、関数のコードの詳しい説明します

関数の実装：説明①

引数（関数に入力される値が、代入されている変数のこと）

関数名

「：」を忘れずに！

```
def calc_hypotenuse(x,y):
    ans = (x**2 + y**2)**(1/2)
    return ans
```

処理

関数の実装：説明②

```
def calc_hypotenuse(x,y):
    ans = (x**2 + y**2)**(1/2)
    return ans
```

返り値：関数から出力される値のこと
「return 出力したい値」と書く

インデント！

これが基本的な実装方法。でも実はプログラミングの関数には、数学の関数にはない特殊な所があるよ

プログラミングの関数の特殊性

プログラミングにおける関数の特殊性：入力や出力がなくても呼び出せる

つまり、引数やreturnがなくてもOK

```
def print_func(): # 関数の定義
    print("関数が呼ばれました")
print_func() # 関数を呼び出す
```

```
関数が呼ばれました
```

【演習】関数

問題１　身長と体重を入力してBMIの値を出力する関数を作成せよ

ただしBMIは　体重 [kg]/(身長 [m]×身長 [m])　で与えられる

問題２　身長と体重を入力としBMIの値を元に肥満度を出力する関数を作成せよ

BMIと肥満度の関係は右表の通りとする

BMI	肥満度
１８未満	“痩せ気味”
１８以上２５未満	“普通”
２５以上	“太り気味”

問題３　問題２で作った関数に以下のデータを入力して結果を確認せよ

tanaka = {“height”:2 , “weight”:80}

【解答】関数

問題 1

```
def BMI_function(height, weight): # BMI_function(height, weight)の定義
    ans = weight / (height * height) # BMIの計算
    return ans # ansが関数の戻り値
```

【解答】関数

問題２

```
def check_obesity(height, weight):
    bmi = BMI_function(height, weight)

    if bmi <18: #BMIの値が18より下であればif文の処理を実行。
        return "痩せ気味"
    elif bmi >= 18 and bmi < 25: #elifでさらに条件式と比較する
        return "普通"
    else: #全て偽であれば以下を実行
        return "太り気味"
```

【解答】関数

問題３

```
Tanaka = {"height":2, "weight":80}
degree_obesity = check_obesity(Tanaka["height"], Tanaka["weight"])
print(degree_obesity) # 肥満度を出力

# 以下は参考:BMIの値を出力する
#BMI_function()を呼び出し、関数の戻り値を変数BMIに代入される。
BMI = BMI_function(Tanaka["height"], Tanaka["weight"])
print(BMI) # BMI値を出力
```

便利な組み込み関数

- 組み込み関数：pythonには標準で備わっている便利な関数
- 使用頻度の多い以下を紹介
  - range()
  - format()

range()

これまで学んだ知識で、１から１０までの整数を標準出力するには？

range()

これまで学んだ知識で、１から１０までの整数を標準出力するには？

```
＃リストを使って1～10までの整数を標準出力する
for i in [1,2,3,4,5,6,7,8,9,10]:
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

[1,2,3,4,5,6,7,8,9,10]

これは面倒！

range()を使う！

range()

range([開始,] 終了[, 刻み])：数列を出力

① range(5)：0から順に４まで

② range(2, 5)：２から順に４まで出力

③ range(2, 5, 2)：２から４まで２刻みで出力

①
```
for i in range(5):
    print(i)
```
```
0
1
2
3
4
```

②
```
for i in range(2,5):
    print(i)
```
```
2
3
4
```

③
```
for i in range(2,5,2):
    print(i)
```
```
2
4
```

終了の値は含まないことに注意！

format()

まずは以前実装したcalc_hypotenuseを用いて、

底辺が5、高さが12の三角形の斜辺の長さを求めよう

format()

まずは以前実装したcalc_hypotenuseを用いて、

底辺が5、高さが12の三角形の斜辺の長さを求めよう

```
# 以前実装したcalc_hypotenuseを用いて、底辺が5、高さが12の三角形の斜辺の長さを求める
ans = calc_hypotenuse(5,12)
print(ans)
```

```
13.0
```

標準出力の形式を “斜辺の長さは13.00” にするにはどうする？

format()

“斜辺の長さは{:.2f}".format(ans)

ansの中身を小数２桁のfloat形式で文字列に埋め込む

```
# "斜辺の長さは13.00"という形で標準出力したい
print("斜辺の長さは{:.2f}".format(ans)) # format関数を用いた方法
```

```
斜辺の長さは13.00
```

もう１つ方法あり

fstring

f"斜辺の長さは{ans:.2f}"

ansの中身を小数２桁のfloat形式で文字列に埋め込む

```
print(f"斜辺の長さは{ans:.2f}") # fstringを用いた方法
```

```
斜辺の長さは13.00
```

formatとfstring

複数の値を埋め込むことも可能

```
name = "Taro"
age = 20
# 複数の値を文字列に埋め込むことも可能
print("私の名前は{}です。年齢は{:d}歳です。".format(name, age)) # format()
print(f"私の名前は{name}です。年齢は{age:d}歳です。") # fstring
```

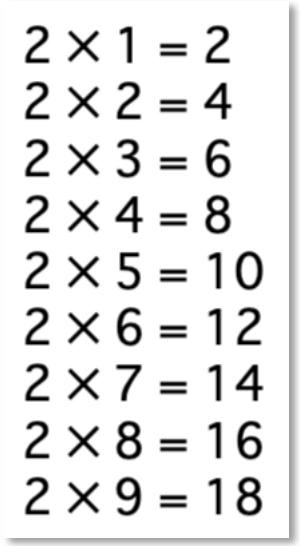
```
私の名前はTaroです。年齢は20歳です。
私の名前はTaroです。年齢は20歳です。
```

【演習】関数

問題４

九九の二の段を以下のような形式で標準出力せよ

ただしrange関数を用いよ



【解答】関数

問題４

```
# format()を用いた場合
base = 2
for i in range(1,10):
    print("{:d} × {:d} = {:d}".format(base, i, base*i))
```

```
2 × 1 = 2
2 × 2 = 4
2 × 3 = 6
2 × 4 = 8
2 × 5 = 10
2 × 6 = 12
2 × 7 = 14
2 × 8 = 16
2 × 9 = 18
```

```
# fstringを用いた場合
base = 2
for i in range(1,10):
    print(f"{base} × {i} = {base*i}")
```

```
2 × 1 = 2
2 × 2 = 4
2 × 3 = 6
2 × 4 = 8
2 × 5 = 10
2 × 6 = 12
2 × 7 = 14
2 × 8 = 16
2 × 9 = 18
```

クラス

超重要です！！

これらを実行するためには

「クラス」を使う必要がある！

クラスとは？

- ソフトウェア工学における「オブジェクト指向」という概念で出てくる言葉
- この概念そのもの理解することは非常に難しい*

クラスとは何か？
どう使うのか？

機械学習を含むデータサイエンスでは、
この２つを理解するだけで十分！

*Javaなどのオブジェクト指向の色が強い言語で
アプリケーションを作るときには理解必須となる

クラスとは？

もしあなたが「車」を作るとなったらどうしますか？

基本となるモデルAを作った後、よりスピードの出るモデルBを作ることも決定しています

クラスとは？

もしあなたが「車」を作るとなったらどうしますか？

基本となるモデルAを作った後、よりスピードの出るモデルBを作ることも決定しています

まずはモデルAの設計図を作り、それからやっとモデルAを作るのでは？

クラス

モデルBの設計図は、モデルAの設計書をベースに作るのでは？

クラスとは？

クラス：設計図

プログラミングにおける設計図に必要なもの

状態を表す変数

機能を表す関数

例えば車だったら？

状態	ガソリンの量 速度
機能	走る 旋回する

クラスとは？

クラス：設計図

プログラミングにおける設計図に必要なもの

状態を表す変数

機能を表す関数

例えば車だったら？

状態	ガソリンの量 速度
機能	走る 空調をかける

クラスの書き方

```
class ModelA:

    def __init__(self):
        self.gas = 100

    def run(self):
        self.gas = self.gas - 5
        print("10km走りました")

    def charge(self, supply_amount):
        self.gas = self.gas + supply_amount
        print("現在のガソリンは{}リットルです".format(self.gas))
```

状態：

- 正式にはフィールド（メンバ変数）
- 実体ごとに異なる値をとってもいい

機能：

- 正式にはメソッド
- すべての実体が共通して持っている

init・・・とは何？　　self・・・とは何？

クラスの書き方

__init__

- コンストラクタと呼ぶ
- クラスが持つフィールドを定義する
- init をアンダーバー（ _ ）２つで挟んでいます

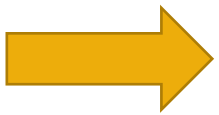
self

- 「クラスに属している」ということを明示するために付ける
- 「フィールドの先頭」と「関数の第一引数」にはselfをつける
- クラス内からフィールドへアクセスする場合にはselfが必要

実体（インスタンス）の生成

クラスを作成した

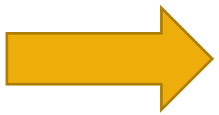
実体を作ろう！



実体（インスタンス）の生成

クラスを作成した

実体を作ろう！



```
model_A = ModelA()
```

実体のことをインスタンスと呼ぶ

実体（インスタンス）の生成

ドット「.」演算子でメソッドとフィールドを呼べる！

```
model_A.run()
```

```
10km走りました
```

```
print(model_A.gas)
```

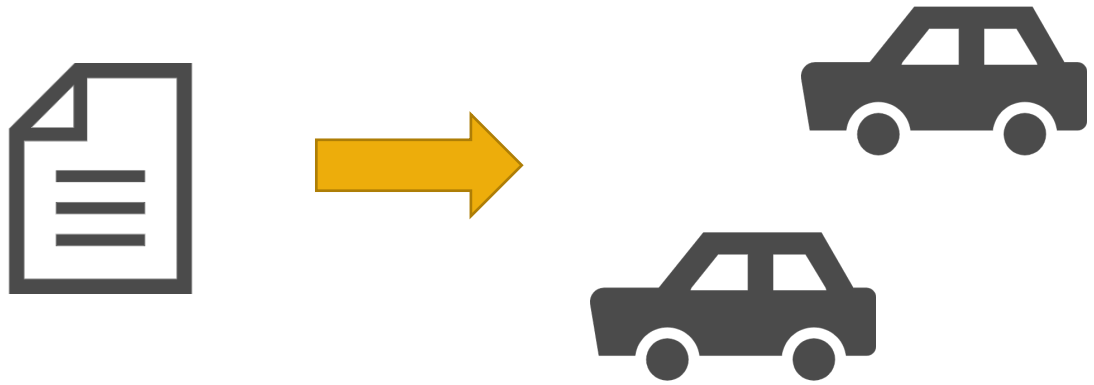
```
95
```

クラスで定義したrun()メソッドの第一引数selfは無視して良い

実体（インスタンス）の生成

インスタンスは複数作っても問題ない！

```
model_A_2 = ModelA()
model_A_3 = ModelA()
```



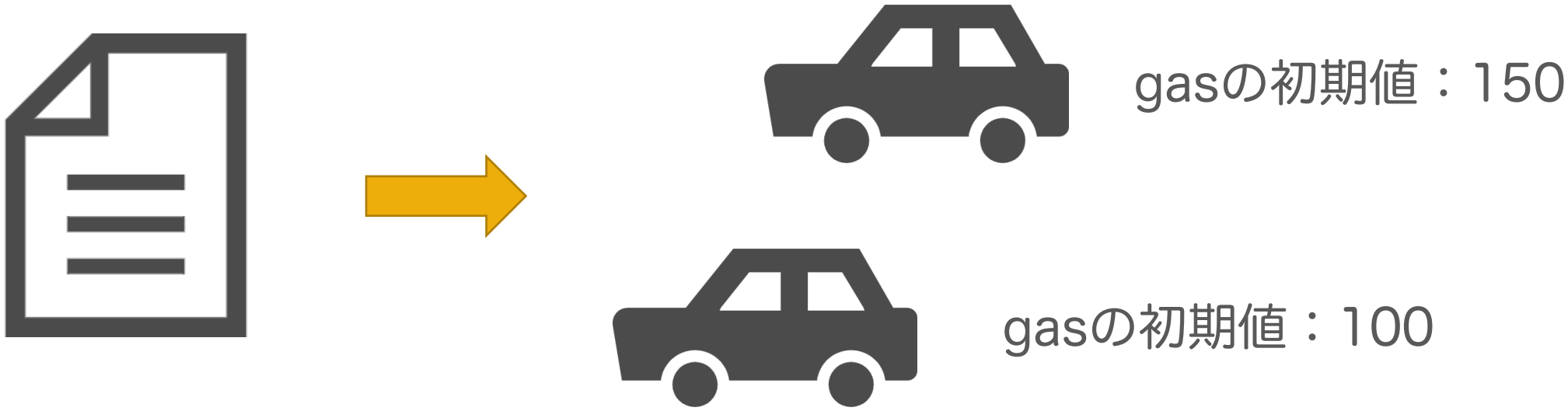
ゲームを作るときには非常に便利

- テトリスのブロック
- シューティングゲームの敵

状態と機能に当たるものは何でしょう？

初期値の異なる実体を作りたい場合

こんなときどうする？



設計図に少しだけ細工をします！

初期値の異なる実体を作りたい場合

```
class ModelA_2:

    def __init__(self, gas):  # __init__の引数にgasを追加
        self.gas = gas # gasの値が self.gasの初期値になる

    def run(self):
        self.gas = self.gas - 5
        print("10km走りました")

    def charge(self, supply_amount):
        self.gas = self.gas + supply_amount
        print("現在のガソリンは{}リットルです".format(self.gas))
```

先程のModelAとの違いに注目！

__init__の引数が増えています！

```
model_A_2 = ModelA_2(200)
model_A_2.gas
```

```
200
```

実体生成のときには、

クラスの引数に初期値を渡す！

重要なので一旦まとめ！

クラスとは？

クラスの構成要素２つは？

インスタンスとは？

インスタンスに「.」演算子を使うと何ができる？

重要なので一旦まとめ！

クラスとは？　　設計図のこと

クラスの構成要素２つは？　　フィールドとメソッド

インスタンスとは？　　クラスから作られた実体

インスタンスに「.」演算子を使うと何ができる？

フィールドとメソッドを呼び出せる

機械学習におけるクラスの使い道

機械学習においてこのクラスをどのようなタイミングで使うでしょうか！？

機械学習におけるクラスの使い道

機械学習においてこのクラスをどのようなタイミングで使うでしょうか！？

モデルの生成	model = tree.DecisionTreeClassifier()
モデルの学習	model.fit(tr_train_X, tr_train_Y)
モデルの予測	predict = model.predict(tr_test_X)

第６章「機械学習モデルの構築と評価」をお楽しみに！

【演習】クラス

問題１

右表のフィールド

下表のメソッド

を持ったHumanクラスを作成せよ

フィールド	意味	初期値
health	体力	１００
time	持ち時間	１０
achieve	業績	０

メソッド	詳細
work： 体力を消費して業績を伸ばす	体力を「５０」持ち時間を「１」消費し 業績を「１」増やす
rest： 持ち時間を消費して体力を回復する	持ち時間を「３」消費し、 体力を「５０」増やす

【解答】クラス

問題１

```
class Human: # Humanクラスを宣言
    def __init__(self): # Humanクラスが呼び出されるときに実行される
        print("initialized")
        self.health = 100 # インスタンスがもつhealthを100に初期化
        self.time = 10 # インスタンスがもつtimeを10に初期化
        self.achieve = 0 # インスタンスがもつメソッドを0に初期化

    def work(self):
        print("work")
        self.achieve += 1
        self.time -= 1
        self.health -= 50

    def rest(self):
        print("rest")
        self.time -= 3
        self.health += 50
```

【解答】クラス

問題１

```
human1 = Human() # human1という名前のインスタンスを生成
print("health: {}".format(human1.health))
print("time: {}".format(human1.time))
print("achieve: {}".format(human1.achieve))
```

```
initialized
health: 100
time: 10
achieve: 0
```

```
human1.work()
print("health: {}".format(human1.health))
print("time: {}".format(human1.time))
print("achieve: {}".format(human1.achieve))
```

```
work
health: 50
time: 9
achieve: 1
```

```
human1.rest()
print("health: {}".format(human1.health))
print("time: {}".format(human1.time))
print("achieve: {}".format(human1.achieve))
```

```
rest
health: 100
time: 6
achieve: 1
```

クラスの継承

モデルAにターボ機能を付け加えたモデルBを作りたい

モデルAの設計図

モデルA

モデルB

０から設計図を作り直す？

クラスの継承

モデルAにターボ機能を付け加えたモデルBを作りたい

モデルAの設計図

モデルA

モデルAに追加機能を付け加えた設計図で十分！

モデルB

クラスの継承

クラスの継承：あるクラスの情報を受け継ぐ

- ModelA：親クラス
- ModelB：子クラス

```
class ModelB(ModelA): # ModelAクラスを継承
    def turbo(self): #ターボを再現する関数
        self.gas = self.gas-10 #ガソリンを10減らす代わりにターボを発動
        print("ターボ機能を発動します！")
```

明記されてなくてもModelBはModelAのフィールドとメソッドを持っている

それを確認するためのコードを考えてみよう！

クラスの継承

```
model_B = ModelB() #model_Bという名前のインスタンスを生成
model_B.run() # ModelAのrun()を受け継いでいる
model_B.turbo()
print(model_B.gas)
```

```
10km走りました
ターボ機能を発動します！
85
```

modelAクラスで定義されていた

run()とgasが呼び出せていますね！

機械学習における継承の使い道

- 色々なモデルを組み合わせた機械学習モデルを作りたい
- scikit-learnの便利な関数を使いつつ、新たなモデルを作成したい

例

```
from sklearn.base import BaseEstimator, ClassifierMixin
class MeanClassifier(BaseEstimator, ClassifierMixin):
```

※今回の講座では利用しません

【演習】クラス

問題２

Humanクラスを継承したEngineerクラスを定義せよ

Engineerクラスは以下のメソッドを追加で持つ

メソッド	詳細
work_hard： 体力を多く消費して業績を大きく伸ばす	体力を「１５０」持ち時間を３消費し 業績を「５」増やす

【解答】クラス

問題２

```
class Engineer(Human):
    def work_hard(self):
        print("work hard")
        self.achieve += 5
        self.health -= 150
        self.time -= 3
```

```
engineer = Engineer()
engineer.rest()
engineer.rest()
engineer.work_hard()
print("health: {}".format(engineer.health))
print("time: {}".format(engineer.time))
print("achieve: {}".format(engineer.achieve))
```

```
initialized
rest
rest
work hard
health: 50
time: 1
achieve: 5
```

Day2で学んだことを確認

- 関数を使うと何が嬉しい？

- クラスを一言で言うと？

Day2で学んだことを確認

- 関数を使うと何が嬉しい？
  - 機能を表すコードを何度も記述しなくて済む
- クラスを一言で言うと？
  - 設計図

お疲れ様でした！

確認テストを解きましょう！