金沢大学
KANAZAWA UNIVERSITY

Pythonを学ぶ上で避けられない「オブジェクト」とは？

金沢大学　軸屋一郎

Supporting materials

YouTube

2023年8月19日

動機

高校教員向けですが
高校生が学ぶのにも
実は良い素材です

高等学校情報科「情報Ⅰ」教員研修用教材（本編）

第3章コンピュータとプログラミング　123ページ

■乱数を用いたプログラムの例

ある値 a があるときに，0 ～ 9 までの数をランダムに発生させる乱数 r と比較して，大きい場合に「a の方が大きい」，小さい場合に「a の方が小さい」，等しい場合に「当たり」と表示するプログラムを図表 4 に示す。

Python で乱数を扱う場合は random モジュールを import 文で読み込む必要があり，「random.randrange(10)」で 0 ～ 9 の 10 個の整数をランダムに発生させる乱数として表現している。

```
1  import random                    #random モジュールを読み込む
2  a = 5
3  r = random.randrange(10)         #0 ～ 9 までの整数をランダムに発生
4  if a==r:
5      print(" 当たり ")
6  elif a>r:
7      print("a の方が大きい ")
8  elif a<r:
9      print("a の方が小さい ")
```

importとは？

ドットの意味は？

図表 4　乱数を用いたプログラムの例 1

https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1416756.htm

Pythonプログラムの**おまじない**を解明したい！

真の**課題解決能力**が
身に付きます

概要

- [教材の概要](#) 6:40
- [Python環境整備と動作確認](#) 5:33
  - [Pythonの基本１](#) 30:16
- オブジェクトの理解
  - [概念](#) 15:24
  - [例示](#) 40:16
  - [Pythonの基本２](#) 23:21
  - [クラスの一般的表記](#) 21:11
- 外部パッケージの活用
  - [モジュール](#) 22:33
  - [パッケージ](#) 17:57
  - [ドットの謎](#) 6:16
- [まとめ](#) 2:58

	高校情報	AIドローン講義	Python経験者
Python環境整備と動作確認	○	◎	△
Pythonの基本１	◎	△	○
概念	△	◎	○
例示	△	△	○
Pythonの基本２	◎	△	○
クラスの一般的表記			◎
モジュール		△	◎
パッケージ		△	◎
ドットの謎		△	○
まとめ			○

自分でプログラムを書いて動作させる経験が大事です

AIドローン講義ではThonnyを使用します

イラストを見てイメージをつかみ、プログラムを動かして具現化できることを目指してます

フローチャートを眺めるだけでも良いです

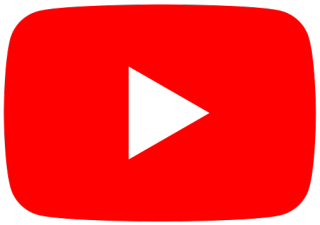
インスタンスメソッドとクラスメソッドの違いを判別できるなら見なくて良いです

イラストを眺めるだけでも良いです

モジュールとパッケージを理解しているなら見なくて良いです

概要

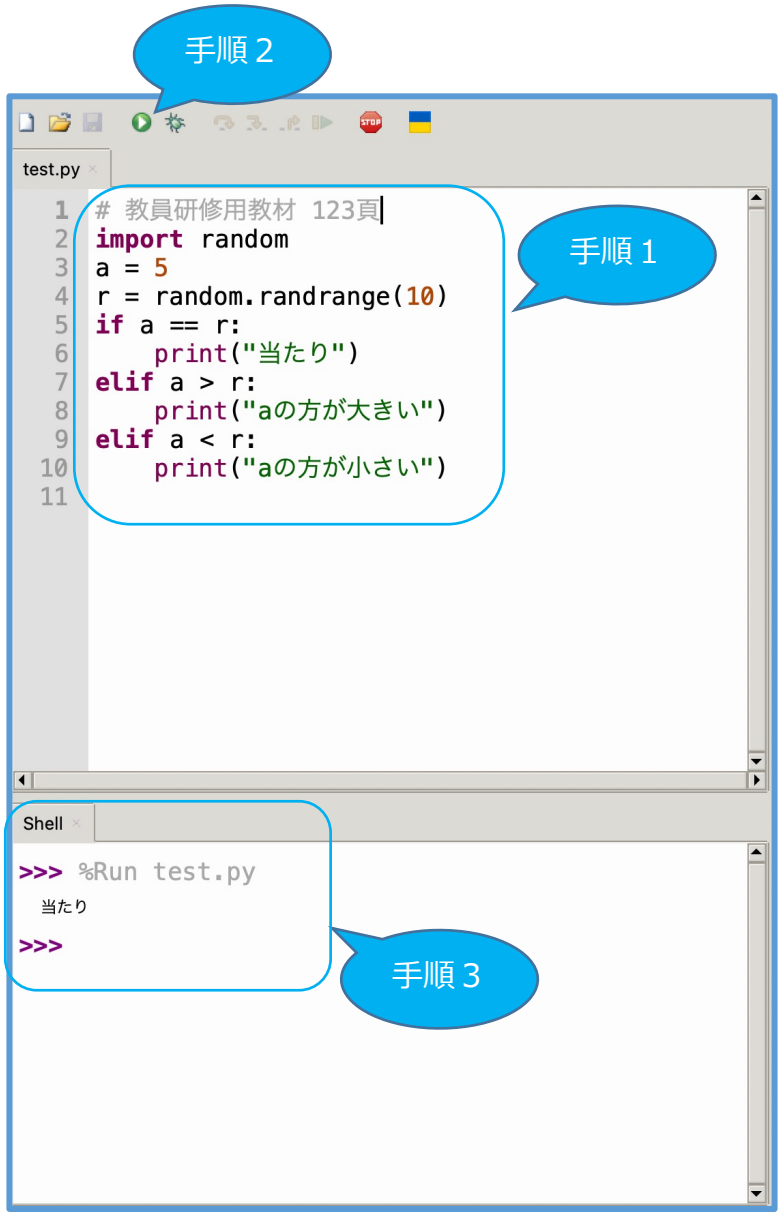
5:33



- **Python環境整備と動作確認**
  - Pythonの基本１
- オブジェクトの理解
  - 概念
  - 例示
  - Pythonの基本２
  - クラスの一般的表記
- 外部パッケージの活用
  - モジュールとパッケージ
  - ドットの謎
- まとめ

プログラミング体験が
課題解決の第一歩

Python実行環境の構築（オフライン）



- ✓ Windows・Macにてオフライン実行するなら Thonny がお勧め.
  - ✓ 公式ページからインストーラをダウンロードして簡単にインストールできます.
  - ✓ **Pythonファイル**を新規作成して実行・保存を行う.
  - ✓ Pythonファイルの**拡張子**は **.py**.

Thonnyの実行画面

手順１：画面上部にスクリプトを記載

手順２：**実行（緑色三角ボタンを押す）**

手順３：画面下部に実行結果を表示

Python実行環境の構築（オンライン）

Windows・Mac・Chromebookにてオンライン実行するなら[Google Colaboratory](#) がお勧め.

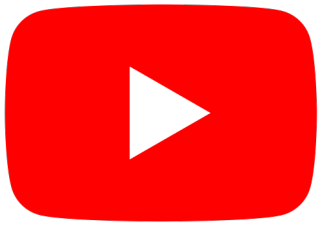
Googleアカウントにログインした状態でGoogle Colaboratoryを開く.

ノートブックを新規作成して実行・保存を行う．ノートブックファイルの**拡張子**は **.ipynb**.

以降はThonnyの実行画面を用いて説明します.

概要

30:16



- Python環境整備と動作確認
  - **Pythonの基本１**
- オブジェクトの理解
  - 概念
  - 例示
  - Pythonの基本２
  - クラスの一般的表記
- 外部パッケージの活用
  - モジュール
  - パッケージ
  - ドットの謎
- まとめ

オブジェクトの理解に必要な
Pythonの基本を学びます

プログラムの動作確認
をする際はビデオ再生
を一時停止して下さい

課題 1-1:

test.py ×

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  if a == r:
6      print("当たり")
7  elif a > r:
8      print("aの方が大きい")
9  elif a < r:
10     print("aの方が小さい")
11
```

条件部末尾はコロンを入れる.
実行部はインデントを入れる.

プログラム本文は半角文字により記載する.
全角文字は使えないので，全角文字の混入を避けて下さい.
Print関数にて文字を出力するときは全角文字も使用可能であり，
引用符（シングルクオーションもしくはダブルクオーテーション）
で囲む.
文頭の # はコメントアウトを表す．転記は不要.

Shell ×

```
>>> %Run test.py
  aの方が大きい
>>>
```

Pythonプログラミングに不慣れな場合は転記して実際に動作検証してみて下さい

課題 1-1：

スクリプトの動作検証をして下さい.

補足：

- 後述の課題に繋げるために適当なファイル名（例えば test.py）をつけて保存して下さい.

結果 1-1:

test.py ×

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  if a == r:
6      print("当たり")
7  elif a > r:
8      print("aの方が大きい")
9  elif a < r:
10     print("aの方が小さい")
11
```

Shell

```
>>> %Run test.py
  aの方が大きい
>>>
```

実行結果：プログラム実行毎に毎回変化

課題 1-1：

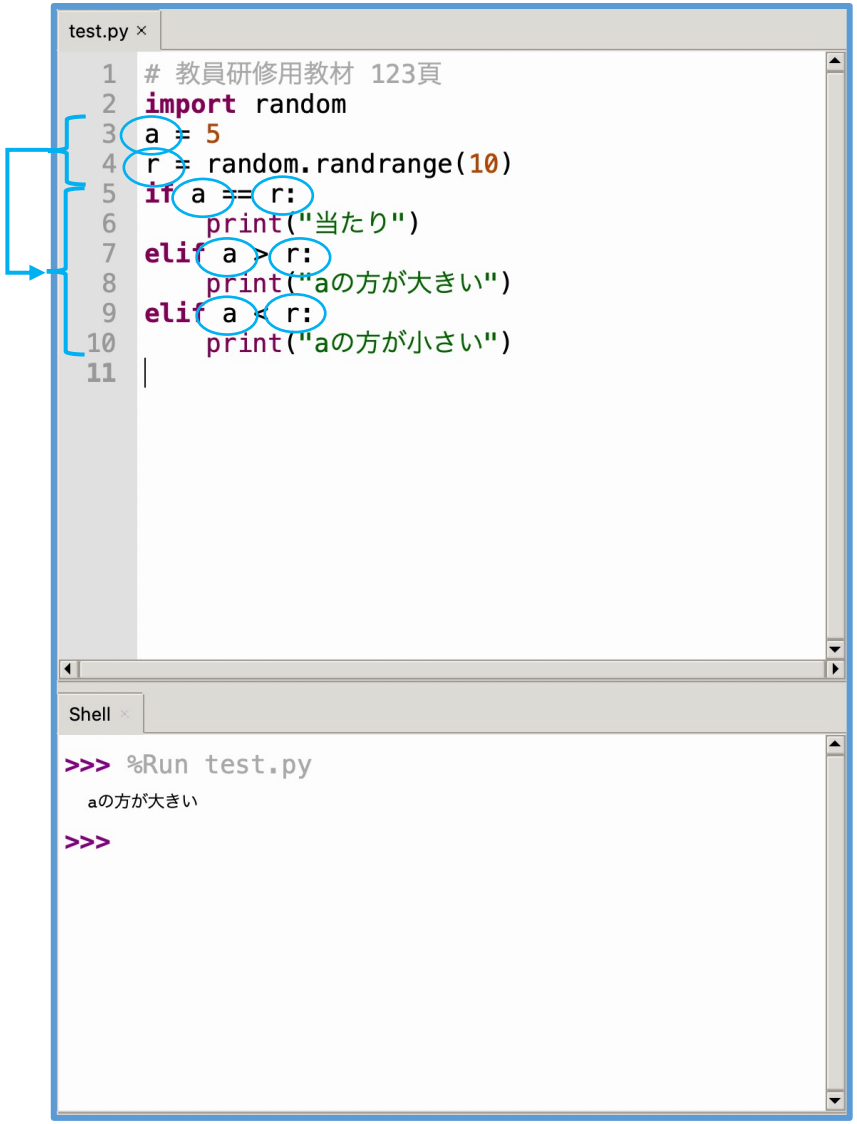
スクリプトの動作検証をして下さい.

結果1-1：

プログラムの動作結果はShellに出力される.

実行結果が毎回変化する.

変数



変数は**データ（オブジェクト）に付けられた名前**を意味します

- 3,4行目では変数 a と変数 r を定義しています
- 5,7,9行目では変数 a と変数 r を活用しています

✓ 数学の変数は「未知もしくは不定な数を表す文字列」を意味します。一般に，Pythonの変数を「データを入れる箱」で例えて説明することが多いのですが，辻褄の合わない事例があり，適切ではありません。Pythonの変数はデータに付けられた名前と解釈すると辻褄が合います。

課題 1-2:

test.py ×

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  if a == r:
6      print("当たり")
7  elif a > r:
8      print("aの方が大きい")
9  elif a < r:
10     print("aの方が小さい")
11
```

Shell

```
>>> %Run test.py
  aの方が大きい
>>>
```

疑問：

変数 r の値は毎回変化するが，値を確認可能？

課題 1-2：

課題 1-1 のプログラムにおいて，変数 r の値を画面出力できるよう，プログラム中の適切な箇所にプリント関数 print('r=', r) を挿入して下さい.

Print関数の使用例：
print('r=') と記載すると文字列「 r= 」を画面出力します.
print(r) と記載すると変数 r の値を画面出力します.
'r=' と r をカンマで結ぶことにより並べて画面出力できます.

結果 1-2:

追加箇所

test.py

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  print('r =', r) # 値を確認
6  print() # 空白行を追加
7  if a == r:
8      print("当たり")
9  elif a > r:
10     print("aの方が大きい")
11 elif a < r:
12     print("aの方が小さい")
13
```

Shell

```
>>> %Run test.py
r = 9
aの方が小さい
>>>
```

実行結果：プログラム実行毎に毎回変化

課題 1-2：

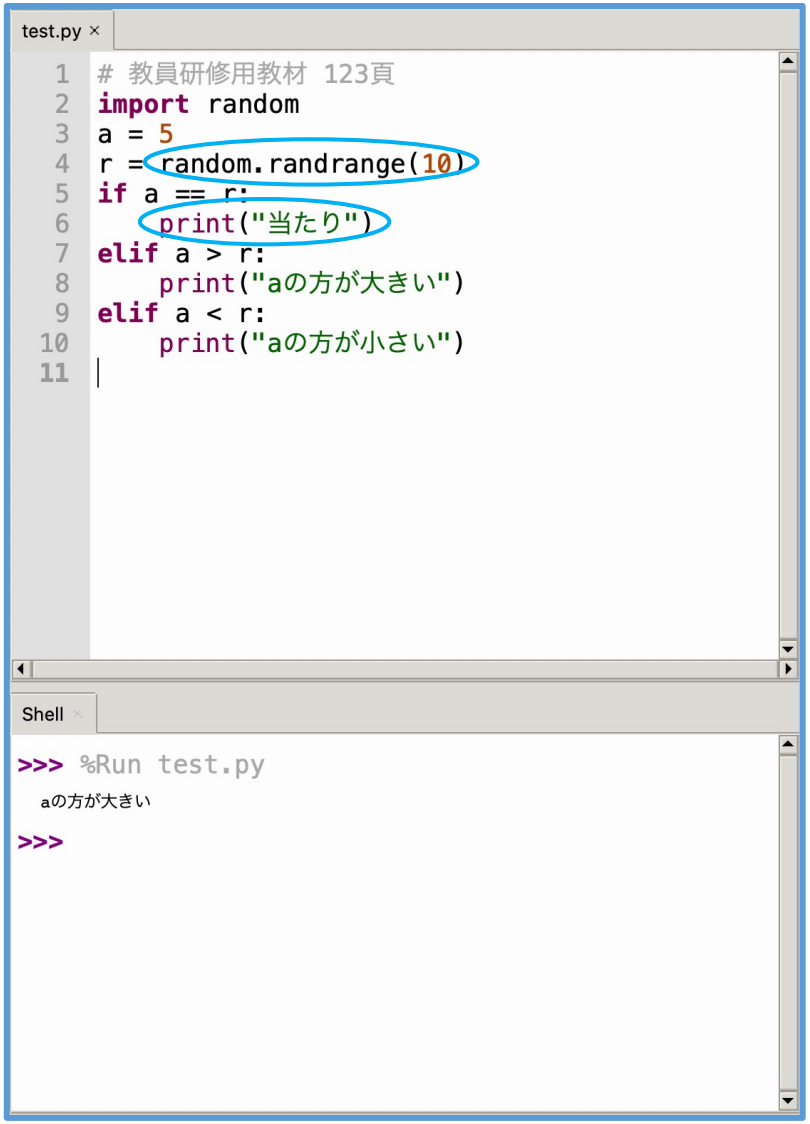
課題 1-1 のプログラムにおいて，変数 r の値を画面出力できるようプリント関数を挿入して下さい．

結果 1-2：

5行目により変数 r の値を画面出力でき，試行毎に値が変化することを確認できる．試行を何回も繰り返すと，関数 random.randrange(10) は 0 から 9 の範囲の整数値乱数を生成することを確認できる．

6 行目は画面出力に空白行を生成することを意図しています．必須ではありません．

関数



関数は**プログラム上の操作**を意味します

- 4行目の random.randrange(n) は関数であり，nを引数として，0からn-1の間の整数値乱数を戻り値として返します
- 6行目の print() は関数であり，引数を画面出力します

✓ 数学の関数は「値から値への写像」を意味します。Pythonの関数は戻り値を返す場合も返さない場合もあり、数学の関数とは異なる働きをすることがわかります。Pythonの関数はプログラム上の操作を意味すると解釈すると辻褄が合います。

✓ 関数の作り方については後述します。

課題 1-3:

test.py

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  print('r =', r) # 値を確認
6  print() # 空白行を追加
7  if a == r:
8      print("当たり")
9  elif a > r:
10     print("aの方が大きい")
11 elif a < r:
12     print("aの方が小さい")
13
```

Shell

```
>>> %Run test.py
  r = 9
  aの方が小さい
>>>
```

疑問：

if文の条件部を構成する論理演算の実体は？

課題 1-3：

if文に入る前にプリント関数

print('a==r', a==r, 'a>r', a>r, 'a<r', a<r)

を追加して，条件部に記載された論理演算 a==r, a>r, a<r をそれぞれ画面出力して下さい.

さらに，プリント関数の出力が7行目以降の実行結果と整合することを確認してください.

結果 1-3:

追加箇所

test.py

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  print('r =', r) # 値を確認
6  print('a==r', a==r, 'a>r', a>r, 'a<r', a<r)
7  print() # 空白行を追加
8  if a == r:
9      print("当たり")
10 elif a > r:
11     print("aの方が大きい")
12 elif a < r:
13     print("aの方が小さい")
14
```

Shell

```
>>> %Run test.py
r = 1
a==r False a>r True a<r False
aの方が大きい
>>>
```

実行結果：プログラム実行毎に毎回変化

課題 1-3：

if文に入る前にプリント関数を追加し，条件部に記載された論理演算 a==r, a>r, a<r をそれぞれ画面出力して下さい.

結果 1-3：

論理演算の結果は**真偽型の真(True)もしくは偽(False)**であることを確認できる.

例えば，r=1 ならば a>r のみが True である.
if文の実行結果は「aの方が大きい」となり，結果が整合することを確認できる.

分岐

test.py

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  print('r =', r) # 値を確認
6  print() # 空白行を追加
7  if a == r:
8      print("当たり")
9  elif a > r:
10     print("aの方が大きい")
11 elif a < r:
12     print("aの方が小さい")
13
```

条件部末尾はコロンを入れる.
実行部はインデントを入れる.

Shell

```
>>> %Run test.py
  r = 9
  aの方が小さい
>>>
```

else if の意味

構造化定理

- **順次：上から順番に実行**
- **分岐：if文を用いると実行を分岐できる**
- **反復：for文を用いると実行を反復できる**

分岐の基本構文

```
if 条件部:
    実行部
```

条件部には論理演算が入る

複数の実行部がある場合はインデントを揃える

分岐を追加

```
if 条件部:
    実行部
elif 条件部:
    実行部
```

```
if 条件部:
    実行部
else:
    実行部
```

※ 動画説明から elif の後に条件部を追加訂正

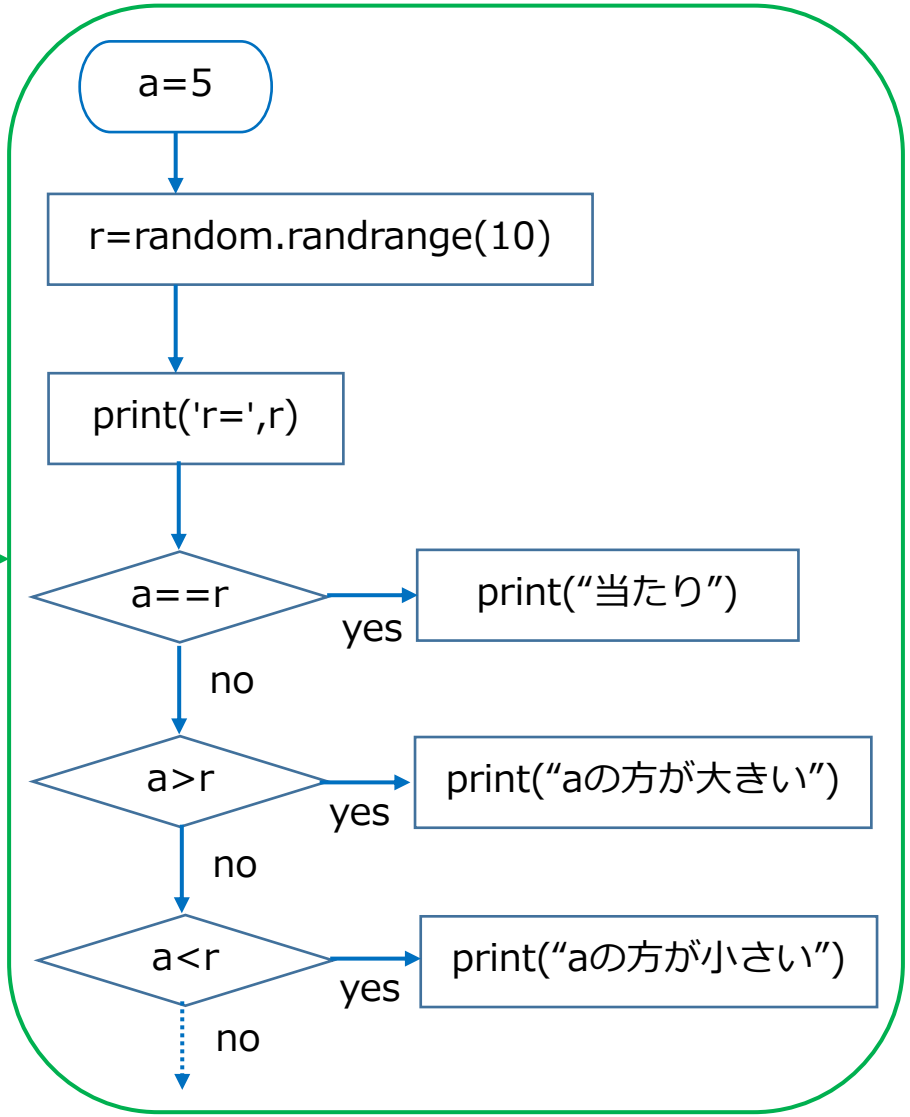
分岐（フローチャート）

test.py

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  print('r =', r) # 値を確認
6  print() # 空白行を追加
7  if a == r:
8      print("当たり")
9  elif a > r:
10     print("aの方が大きい")
11 elif a < r:
12     print("aの方が小さい")
13
```

Shell

```
>>> %Run test.py
  r = 9
  aの方が小さい
>>>
```



補足：最後の no の処理が未定義となるのでプログラムの書き方としては良くない。最後の elif は else と書く方が良い。

課題 1-4:

test.py

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  print('r =', r) # 値を確認
6  print('a==r', a==r, 'a>r', a>r, 'a<r', a<r)
7  print() # 空白行を追加
8  if a == r:
9      print("当たり")
10 elif a > r:
11     print("aの方が大きい")
12 elif a < r:
13     print("aの方が小さい")
14
```

Shell

```
>>> %Run test.py

  r = 1
  a==r False a>r True a<r False

  aの方が大きい

>>>
```

疑問：

変数 r を特徴付けるパラメータは？

課題 1-4：

type()関数とid()関数を用いて変数 r を特徴づけるパラメータを調べます． if文に入る前にプリント関数 print(type(r), id(r)) を追加し，type(r) と id(r) をそれぞれ画面出力して下さい．

さらに，繰り返し実行して結果がどのように変化するか観察してください．

結果 1-4:

追加箇所

test.py

```
1  # 教員研修用教材 123頁
2  import random
3  a = 5
4  r = random.randrange(10)
5  print('r =', r) # 値を確認
6  print('type(r) =', type(r)) # 型を確認
7  print('id(r) =', id(r)) # IDを確認
8  print()
9  print('a==r', a==r, 'a>r', a>r, 'a<r', a<r)
10 print() # 空白行を追加
11 if a == r:
12     print("当たり")
13 elif a > r:
14     print("aの方が大きい")
15 elif a < r:
16     print("aの方が小さい")
17
```

Shell

```
>>> %Run test.py

  r = 1
  type(r) = <class 'int'>
  id(r) = 4523245808

  a==r False a>r True a<r False

  aの方が大きい

>>>
```

試行によらず一定
試行毎に変化

課題 1-4：

type()関数とid()関数を用いて変数 r を特徴づけるパラメータを調べます． If文に入る前にプリント関数 print(type(r), id(r)) を追加し，type(r) と id(r) をそれぞれ画面出力して下さい．

結果 1-4：

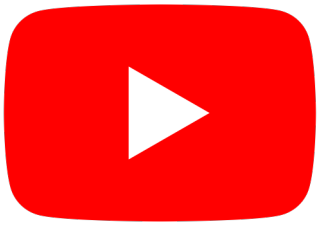
変数 r には値だけでなく，**型(class)とIDが内包**されています．型は整数(int)型であり試行によらず一定です．IDは試行毎に変化する．

疑問：

Pythonでデータを表す枠組みは？

概要

15:24



- Python環境整備と動作確認
  - Pythonの基本１
- **オブジェクトの理解**
  - **概念**
  - 例示
  - Pythonの基本２
  - クラスの一般的表記
- 外部パッケージの活用
  - モジュール
  - パッケージ
  - ドットの謎
- まとめ

車を例にオブジェクトの定義を確認

オブジェクト

- Pythonではデータは**オブジェクト**という抽象的な概念により表現される.

- **オブジェクト**は**値**，**型**，**ID**から構成される.
  - **型**はオブジェクトの設計情報を表す．クラスとかデータ型とも呼ばれる.
  - **ID**はオブジェクトの固有番号を表す.
  - **変数**はオブジェクトの名前を表します.
  - **値**は**メンバ変数**と**メンバ関数**から構成される.
    - **メンバ変数**はオブジェクトに格納されるオブジェクトの名前を表す.
    - **メンバ関数**はオブジェクトに内包された操作を表す.

例え話によりオブジェクトの定義を順番に説明します

車を例としたオブジェクトの理解

車をイメージできますよね？

車両は設計書をもとに製造されています.

設計書

製造

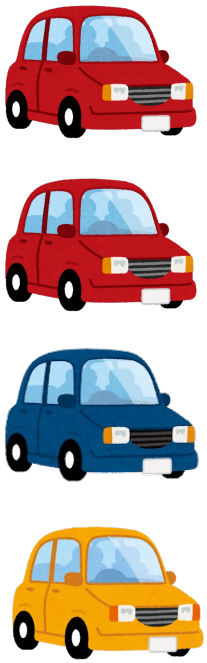
車両

単一の設計書から複数の車両を製造できます．

車両毎に異なる仕様を備えます．

設計書

車両



製造

クラスは設計書に相当します．

オブジェクトは車両に相当します．

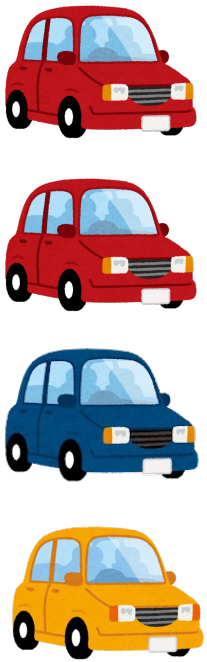
プログラム中でクラスから実体化されたオブジェクトを特に**インスタンス**と呼びます．

（整数や浮動小数点数などPythonに元から組み込まれたオブジェクトも存在します）

クラス

クラスの実体化

オブジェクト （インスタンス）



オブジェクト

- Pythonではデータは**オブジェクト**という抽象的な概念により表現される.

- **オブジェクト**は**値**，**型**，**ID**から構成される.
  - **型**はオブジェクトの設計情報を表す．クラスとかデータ型とも呼ばれる.
  - **ID**はオブジェクトの固有番号を表す.
  - **変数**はオブジェクトの名前を表します.
  - **値**は**メンバ変数**と**メンバ関数**から構成される.
    - **メンバ変数**はオブジェクトに格納されるオブジェクトの名前を表す.
    - **メンバ関数**はオブジェクトに内包された操作を表す.

単一の設計書から複数の車両を製造できます．

車両を区別するために自動車登録番号表（いわゆるナンバー）が割り振られます．

設計書

製造

車両

自動車登録番号標
品川ほ ****

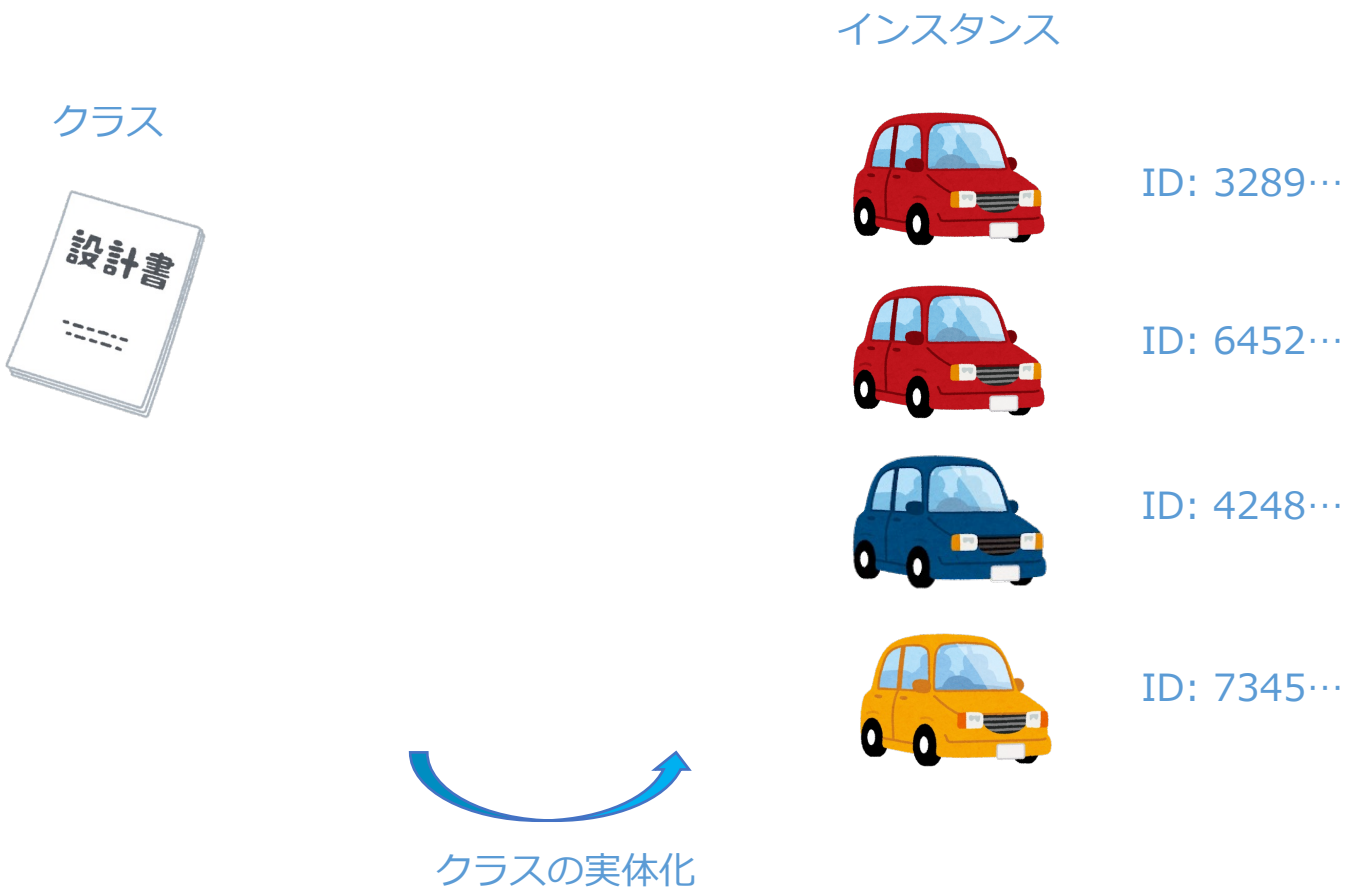
自動車登録番号標
富士山ほ ****

自動車登録番号標
金沢ほ ****

自動車登録番号標
石川ほ ****

IDは自動車登録番号標に相当します．

プログラム実行中のIDは不変な固有番号です．プログラムを再実行するとIDは変化します．強引に例えると，設計書をもとに車を製造し直すことに相当します．



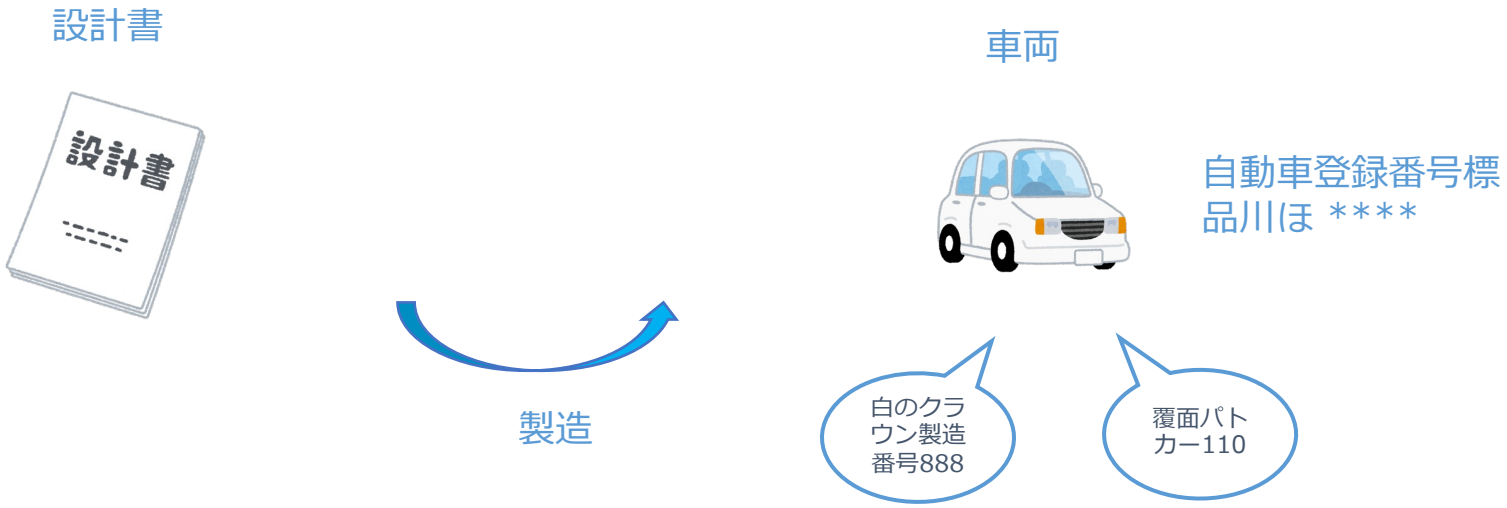
オブジェクト

- Pythonではデータは**オブジェクト**という抽象的な概念により表現される．

- **オブジェクト**は**値**，**型**，**ID**から構成される．
  - **型**はオブジェクトの設計情報を表す．クラスとかデータ型とも呼ばれる．
  - **ID**はオブジェクトの固有番号を表す．
    - プログラム実行時にメモリに確保される領域の先頭アドレスに対応します．
    - プログラム実行中のIDは固定されます．プログラム再実行時にはメモリ領域が確保しなおされるため，IDも変化します．
  - **変数**はオブジェクトの名前を表します．
  - **値**は**メンバ変数**と**メンバ関数**から構成される．
    - **メンバ変数**はオブジェクトに格納されるオブジェクトの名前を表す．
    - **メンバ関数**はオブジェクトに内包された操作を表す．

車両に**名前**をつけることが多いです.

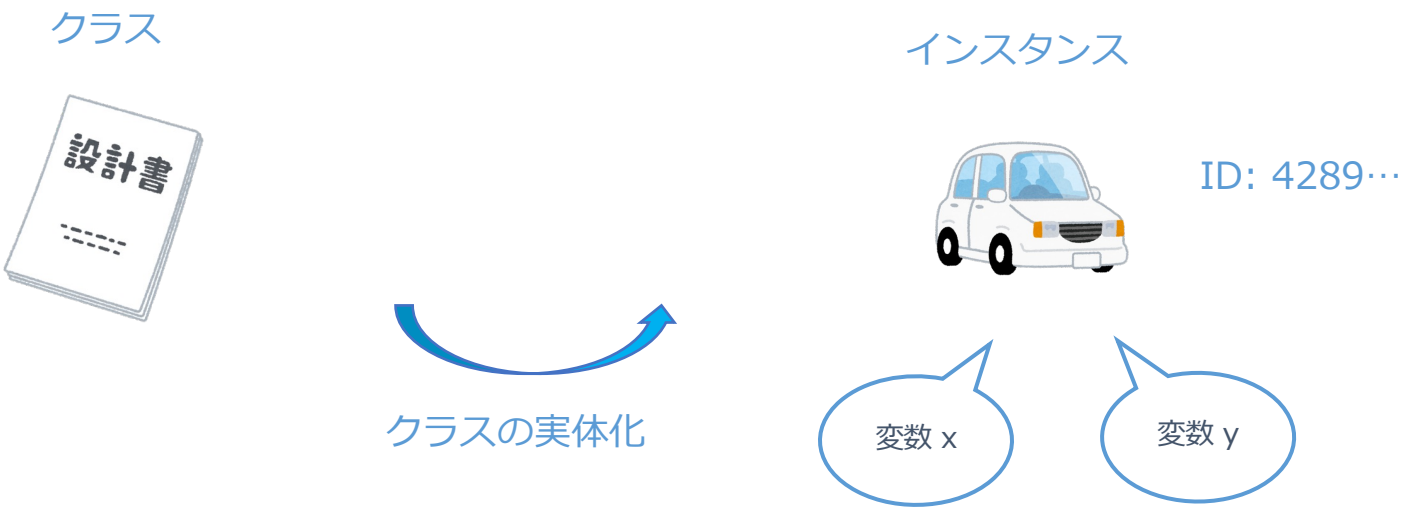
車両に複数の名前をつけることは可能です.



変数はオブジェクトの名前を表します．

一つのオブジェクトに複数の名前を割り振ることも可能です．

Pythonの変数は数学の関数とは少し異なります．変数という名前を冠しますが，数に限定されません．任意のオブジェクトを対象に対して変数名を割り振ることができます．



オブジェクト

- Pythonではデータは**オブジェクト**という抽象的な概念により表現される.

- **オブジェクト**は**値**，**型**，**ID**から構成される.
  - **型**はオブジェクトの設計情報を表す．クラスとかデータ型とも呼ばれる.
  - **ID**はオブジェクトの固有番号を表す.
  - **変数**はオブジェクトの名前を表します.
  - **値**は**メンバ変数**と**メンバ関数**から構成される.
    - **メンバ変数**はオブジェクトに格納されるオブジェクトの名前を表す.
    - **メンバ関数**はオブジェクトに内包された操作を表す.

車には様々な属性と機能があります．

設計書

車両

自動車登録番号標
品川ほ ****

属性に関する設計情報

- 車体カラーを選択可能
- アクセルの傾きは0度から10度の範囲内
- ハンドルの傾きは±540度の範囲内

属性を実現

- 製造時に車体カラーは赤を選択
- アクセルの傾き
- ハンドルの傾き

機能に関する設計情報

- アクセルを踏んだら車が走る
- ハンドルを回すと車の進行方向が変化

機能を実現

- アクセルを踏んだら車が走る
- ハンドルを回すと車の進行方向が変化

メンバ変数は車の**属性**に相当し，**メンバ関数 (メソッド)** は車の**機能**に相当します．

クラス

インスタンス

ID: 4289…

メンバ変数（属性）

属性に関する設計情報
- 車体カラーを選択可能
- アクセルの傾きは0度から10度の範囲内
- ハンドルの傾きは±540度の範囲内

属性を実現
- 製造時に車体カラーは赤を選択
- アクセルの傾き
- ハンドルの傾き

メンバ関数 (メソッド)

機能に関する設計情報
- アクセルを踏んだら車が走る
- ハンドルを回すと車の進行方向が変化

機能を実現
- アクセルを踏んだら車が走る
- ハンドルを回すと車の進行方向が変化

オブジェクト

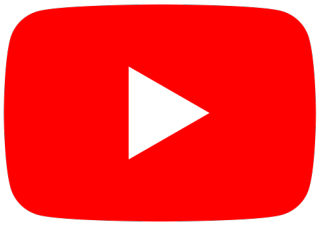
- Pythonではデータは**オブジェクト**という抽象的な概念により表現される.

- **オブジェクト**は**値**，**型**，**ID**から構成される.
  - **型**はオブジェクトの設計情報を表す．クラスとかデータ型とも呼ばれる.
  - **ID**はオブジェクトの固有番号を表す.
  - **変数**はオブジェクトの名前を表します.
  - **値**は**メンバ変数**と**メンバ関数**から構成される.
    - **メンバ変数**はオブジェクトに格納されるオブジェクトの名前を表す.
    - **メンバ関数**はオブジェクトに内包された操作を表す.

補足：　Pythonではメンバ変数を「属性」、メンバ関数を「メソッド」と呼ぶことが多いのですが、意味が分かりづらくなるために、他のプログラミング言語からメンバ変数とメンバ関数の用語を借用して説明しています

概要

40:16



- Python環境整備と動作確認
  - Pythonの基本１
- **オブジェクトの理解**
  - 概念
  - **例示**
  - Pythonの基本２
  - 一般的表記
- 外部パッケージの活用
  - モジュールとパッケージ
  - ドットの謎
- まとめ

サンプルコードの解読
を通じてオブジェクト
を具体的に確認

課題 2-1:（関数を例示）

test.py × testFunc.py

```
1  # 教員研修用教材 123頁 を関数で表現
2  import random
3
4  a = 5
5  r = random.randrange(10)
6
7  def arComp(x,y): # 数値比較部分を関数化
8      if x == y:
9          print("当たり")
10     elif x > y:
11         print(str(x)+"の方が大きい")
12     elif x < y:
13         print(str(x)+"の方が小さい")
14
15 arComp(a,r) # 関数arComp()を適用
16
```

Shell

```
>>> %Run testFunc.py
  5の方が大きい
>>>
```

関数の構文：

```
def 関数名(引数):
    実行部
```

✓ 一連の操作をまとめて関数を構成

✓ 定型化された操作の繰り返しが可能となる

課題 2-1:

スクリプトの動作検証をして下さい.

教員研修用教材123ページのサンプルプログラムと同様の結果が得られることを確認して下さい.

サンプルプログラム（関数を例示）

test.py × testFunc.py

```
1  # 教員研修用教材 123頁 を関数で表現
2  import random
3
4  a = 5
5  r = random.randrange(10)
6
7  def arComp(x,y): # 数値比較部分を関数化
8      if x == y:
9          print("当たり")
10     elif x > y:
11         print(str(x)+"の方が大きい")
12     elif x < y:
13         print(str(x)+"の方が小さい")
14
15 arComp(a,r) # 関数arComp()を適用
16
```

Shell

```
>>> %Run testFunc.py
  5の方が大きい
>>>
```

クラスの説明を行う前に，まず，関数を用いたプログラム作成を例示する．

7行目から13行目は数値比較部分を関数化している．

7行目に関数 arComp() の定義部があり，8行目から13行目が実行部である．

15行目にて関数 arComp() を呼び出し，教員研修用教材123ページのサンプルプログラムと同等の実行結果が得られる．

課題 2-2（クラスを例示）

test.py　testFunc.py　testClass.py

```
1  # 教員研修用教材 123頁 を関数で表現
2  import random
3
4  class randComp:
5      def __init__(self):  __はアンダーラインを重ねている
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC = randComp()
20 rC.arComp()
21
```

Shell

```
>>> %Run testClass.py
  5の方が大きい
>>>
```

クラスの構文：

```
class クラス名:
    def __init__(self, 引数):
        コンストラクタの実行部
    def メソッド(self, 引数):
        メソッドの実行部
    def __del__(self,引数):
        デストラクタの実行部
```

メンバ変数を定義

メンバ関数を定義

✓ オブジェクトの雛形

課題 2-2:　スクリプトの動作検証をして下さい.

補足：　課題 2-2 のサンプルプログラムは保存しておいて下さい．資料後半に使用します。

※ 動画説明からデストラクタの構文を訂正

サンプルプログラム（クラスを例示）

test.py　testFunc.py　testClass.py

```
1  # 教員研修用教材 123頁 を関数で表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC = randComp()
20 rC.arComp()
21
```

Shell

```
>>> %Run testClass.py
  5の方が大きい
>>>
```

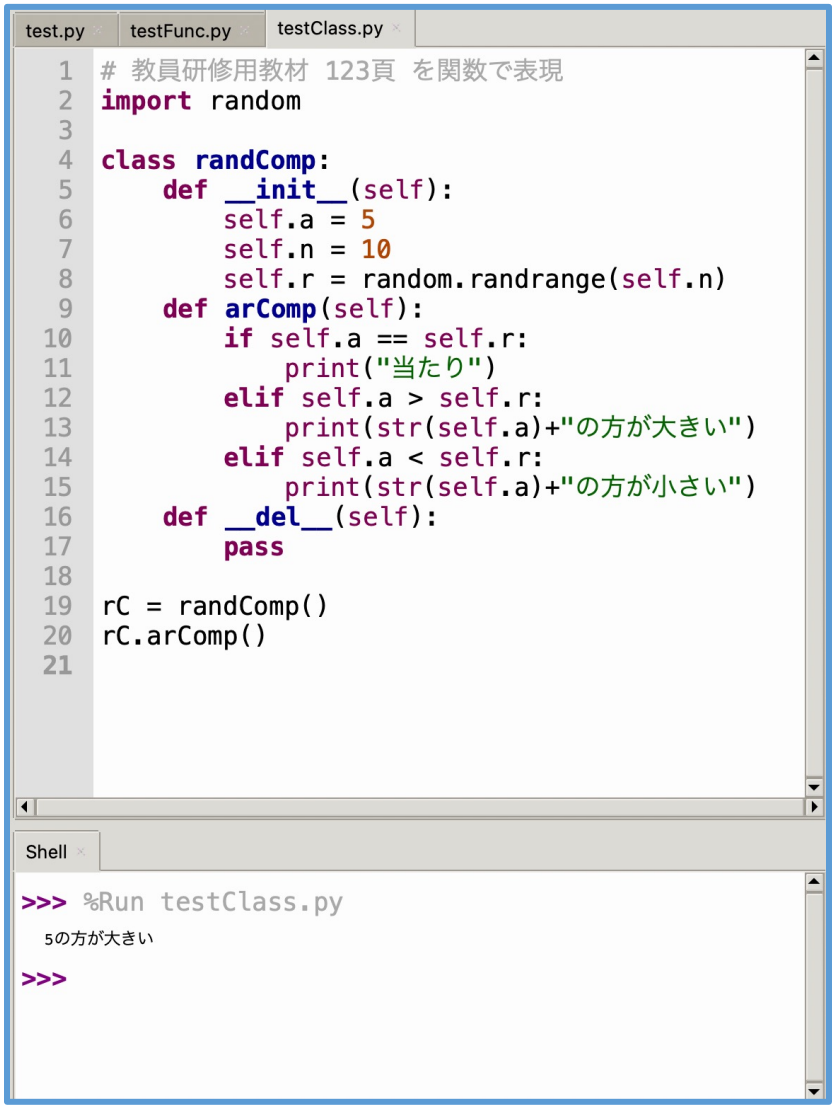
クラスを用いたプログラム作成例を例示する．

プログラム概要：

4行目から17行目にかけてクラス randComp を定義している．

19行目でクラス randComp の初期化を行い，インスタンス rC を生成する．

20行目でメソッド rC.arComp() を呼び出し，教員研修用教材123ページのサンプルプログラムと同等の実行結果が得られる．



詳細解読（クラスの定義方法）：

4行目ではクラスの名称 randComp を定めている．

5行目から8行目にかけて**クラス初期化時の処理**を規定しており，**コンストラクタ**と呼びます．
なお，__init__() はPythonにおける予約語です．

コンストラクタ内部ではインスタンス変数 a, n, r を定義している．ここで self はオブジェクト自身を表す．クラス初期化時にインスタンスが生成され，その際にインスタンスを表す変数名がselfに代入されることを意味する．

ドット「.」によりオブジェクトとメンバ変数の間の階層構造を表す．ここでは「self.a」が「オブジェクト self のメンバ変数 a」を意味する．

test.py | testFunc.py | testClass.py

```
1  # 教員研修用教材 123頁 を関数で表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC = randComp()
20 rC.arComp()
21
```

Shell

```
>>> %Run testClass.py
  5の方が大きい
>>>
```

9行目から15行目にかけてインスタンスメソッド arComp() を定義している．

関数化の際は arComp(x,y) と二個の引数が存在したが，ここでは arComp(self) という形式であり x と y に相当する引数は存在しない．

クラス内部でインスタンス変数を共有できるので，arComp(self) において self 以外の引数を含める必要がないからである．

10行目から15行目の実行部に処理内容が記載されており，教員研修用教材123ページのサンプルプログラムの処理内容と同等である．

16行目から17行目にかけてクラス終了時の処理を記載しており，デストラクタと呼びます．なお，__del__()もPythonにおける予約語です．デストラクタは省略可能です．

test.py testFunc.py testClass.py

```
1  # 教員研修用教材 123頁 を関数で表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC = randComp()
20 rC.arComp()
21
```

Shell

```
>>> %Run testClass.py
  5の方が大きい
>>>
```

詳細解読（インスタンスの定義方法）：

19行目にてクラス randComp からインスタンス rC を生成している.

この時点で5行目から8行目にかけてのコンストラクタ(初期化関数)が呼び出されている.

インスタンス rC にはインスタンス変数 rC.a, rC.n, rC.r が内包され，インスタンスメソッド rC.arComp() を利用可能な状態となっている.

ドット「.」によりオブジェクトとメンバ関数の間の階層構造を表す．ここでは「rC.arComp()」が「オブジェクト rC のメンバ関数 arComp()」を意味する.

test.py | testFunc.py | testClass.py

```
1  # 教員研修用教材 123頁 を関数で表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC = randComp()
20 rC.arComp()
21
```

Shell

```
>>> %Run testClass.py
  5の方が大きい
>>>
```

プログラム20行目の実行結果

詳細解読（インスタンスの活用方法）：

20行目にてインスタンスメソッド rC.arComp() を呼出し，教員研修用教材123ページのサンプルプログラムと同等の実行結果が得られる.

具体的には，rC.a と rC.r の大小に応じて11,13,15行目のいずれかが実行されて画面出力されており，self に rC が代入されていることに注意されたい.

課題 2-3：（オブジェクトの使用例）

test.py　testFunc.py　testClassClass.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC1 = randComp()
20 rC1.arComp()
21
22 rC2 = randComp()
23 rC2.a = 10
24 rC2.arComp()
25
```

課題2-2からの変更箇所

Shell

```
>>> %Run testClassClass.py
  当たり
  10の方が大きい
>>>
```

課題 2-3:

スクリプトの動作検証をして下さい.

スクリプトの全部を転記する必要はなく，課題2-2のスクリプトとの差分を変更すれば良いです.

課題2-2と比較して動作結果がどのように変化するか考察して下さい.

補足：

課題 2-2 のプログラムに上書き保存すると以降で不都合が生じるので，課題 2-2 のプログラムは残して下さい．課題 2-3 のファイルを別に作成して保存すれば良いです.

結果 2-3:

test.py　testFunc.py　testClassClass.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC1 = randComp()
20 rC1.arComp()
21
22 rC2 = randComp()
23 rC2.a = 10
24 rC2.arComp()
25
```

課題2-2からの変更箇所

Shell

```
>>> %Run testClassClass.py
当たり
10の方が大きい
>>>
```

rC1.arComp()の実行結果：変化
rC2.arComp()の実行結果：変化なし

課題 2-3:

スクリプトの動作検証をして下さい.

結果 2-3 :

rC1.arComp() の実行結果は毎回変化する.

rC2.a = 10 の書き換えに伴い必ず self.a<self.r が成立するので，rC2.arComp() の実行結果は「10の方が大きい」 に固定される.

サンプルプログラム（オブジェクトの使用例）

test.py　testFunc.py　testClassClass.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC1 = randComp()
20 rC1.arComp()
21
22 rC2 = randComp()
23 rC2.a = 10
24 rC2.arComp()
25
```

Shell

```
>>> %Run testClassClass.py
当たり                    } rC1.arComp()の実行結果
10の方が大きい            } rC2.arComp()の実行結果
>>>
```

クラスを導入することのご利益を知るために，複数のインスタンス生成を例示する．

19行目ではインスタンス rC1 を生成し，20行目でインスタンスメソッド rC1.arComp() を適用している．プログラムを繰返し実行した時に，判定結果がばらけることが確認できる．

22行目ではインスタンス rC2 を生成し，23行目でインスタンス変数を rC2.a=10 と書換えてから，24行目でインスタンスメソッド rC2.arComp() を適用している．Shellへの出力を確認すると，インスタンス変数の書換えが反映されていることが確認できる．

クラスから複数のインスタンスを独立に生成できたことが確認できる．

課題 2-4：（クラスの改修例）

test.py　testFunc.py　testClassClass.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC1 = randComp()
20 rC1.arComp()
21
22 rC2 = randComp()
23 rC2.a = 10
24 rC2.arComp()
25
```

Shell

```
>>> %Run testClassClass.py

  当たり
  10の方が大きい

>>>
```

課題 2-4:

初期化時にメンバ変数 r の初期値を画面表示できるようにクラス randComp を変更して下さい.

ヒント：

5行目から8行目にかけてのコンストラクタのどこに print(self.r) を追加すればよいでしょうか？

回答 2-4:

test.py　testFunc.py　testClassClass.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9          print(self.r)
10     def arComp(self):
11         if self.a == self.r:
12             print("当たり")
13         elif self.a > self.r:
14             print(str(self.a)+"の方が大きい")
15         elif self.a < self.r:
16             print(str(self.a)+"の方が小さい")
17     def __del__(self):
18         pass
19
20 rC1 = randComp()
21 rC1.arComp()
22
23 rC2 = randComp()
24 rC2.a = 10
25 rC2.arComp()
```

追加箇所

Shell

```
>>> %Run testClassClass.py
9
5の方が小さい
7
10の方が大きい
>>>
```

print(self.r)の実行結果
print(self.r)の実行結果
複数回実行

課題 2-4:

初期化時にメンバ変数 r の初期値を画面表示できるようにクラス randComp を変更して下さい.

回答 2-4:

8行目の下に print(self.r) を挿入.

課題 2-5：（反復・リスト）

test.py　testFunc.py　testClassClass.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9          print(self.r)
10     def arComp(self):
11         if self.a == self.r:
12             print("当たり")
13         elif self.a > self.r:
14             print(str(self.a)+"の方が大きい")
15         elif self.a < self.r:
16             print(str(self.a)+"の方が小さい")
17     def __del__(self):
18         pass
19
20 rC1 = randComp()
21 rC1.arComp()
22
23 rC2 = randComp()
24 rC2.a = 10
25 rC2.arComp()
```

消去して書き換えて下さい

Shell

```
>>> %Run testClassClass.py

  9
  5の方が小さい
  7
  10の方が大きい

>>>
```

課題 2-5:

randComp() を9回繰り返してオブジェクトを上書き生成し，メンバ変数 r をリスト rSave に保存できるようにプログラムを変更.

ヒント：

まず rSave = [] として空のリストを作成します．次に，rSave.append(x) によりリストに変数 x を追加します．ここでの x としてメンバ変数 r を指定します．for文を用いることにより反復できます．

補足：

難しいと感じる場合にはスキップして下さい．For文やリストについては後述します．

回答 2-5:

test.py × testFunc.py testClassClass.py testClassClassEx2.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rSave = []
20
21 for k in range(9):
22     rC = randComp()
23     rSave.append(rC.r)
24
25 print(rSave)
```

変更箇所

Shell

```
>>> %Run -c $EDITOR_CONTENT
  [9, 4, 8, 9, 4, 7, 2, 1, 2]
>>>
```

プログラム25行目の実行結果

課題 2-5:

randComp() を9回繰り返してオブジェクトを上書き生成し，メンバ変数 r をリスト rSave に保存できるようにプログラムを変更.

回答 2-5:

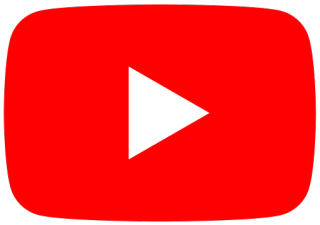
リスト rSave の初期値として空リスト [] を生成する． For文を用いてコンストラクタ randComp() とインスタンス変数 rC.r をリスト rSave に追加する操作を繰り返す.

実行結果は print(rSave) により確認できる.

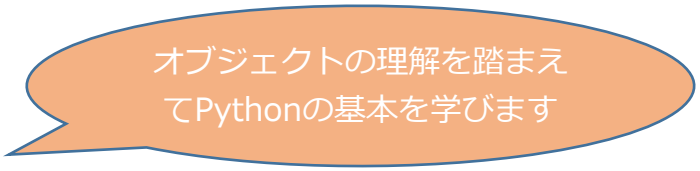
コンストラクタが呼び出されるたびにインスタンスが初期化され，rC.r に異なる乱数が代入されることに注意されたい.

概要

23:21



- Python環境整備と動作確認
  - Pythonの基本１
- **オブジェクトの理解**
  - 概念
  - 例示
  - **Pythonの基本２**
  - クラスの一般的表記
- 外部パッケージの活用
  - モジュール
  - パッケージ
  - ドットの謎
- まとめ



反復

test.py × | testFunc.py | testClassClass.py | testClassClassEx2.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rSave = []
20
21 for k in range(9):
22     rC = randComp()
23     rSave.append(rC.r)
24
25 print(rSave)
```

実行部で変数kは直接的に使用されないが，変数kが0から8まで変化する間に実行部が9回繰り返される

Shell

```
>>> %Run -c $EDITOR_CONTENT
  [9, 4, 8, 9, 4, 7, 2, 1, 2]
>>>
```

プログラム25行目の実行結果

構造化定理

- **順次：上から順番に実行**
- **分岐：if文を用いると実行を分岐できる**
- **反復：for文を用いると実行を反復できる**

反復の基本構文

```
for 変数 in シーケンス:
    実行部
```

シーケンス型：リストなど，複数のデータが順番に並んでいる型

シーケンスの各要素を順番に変数として取り出します。変数を実行部で使用可能ですが必ずしも使用数とは限りません。

For each variable in the sequence,
do the actions.
と書くと解釈しやすいです

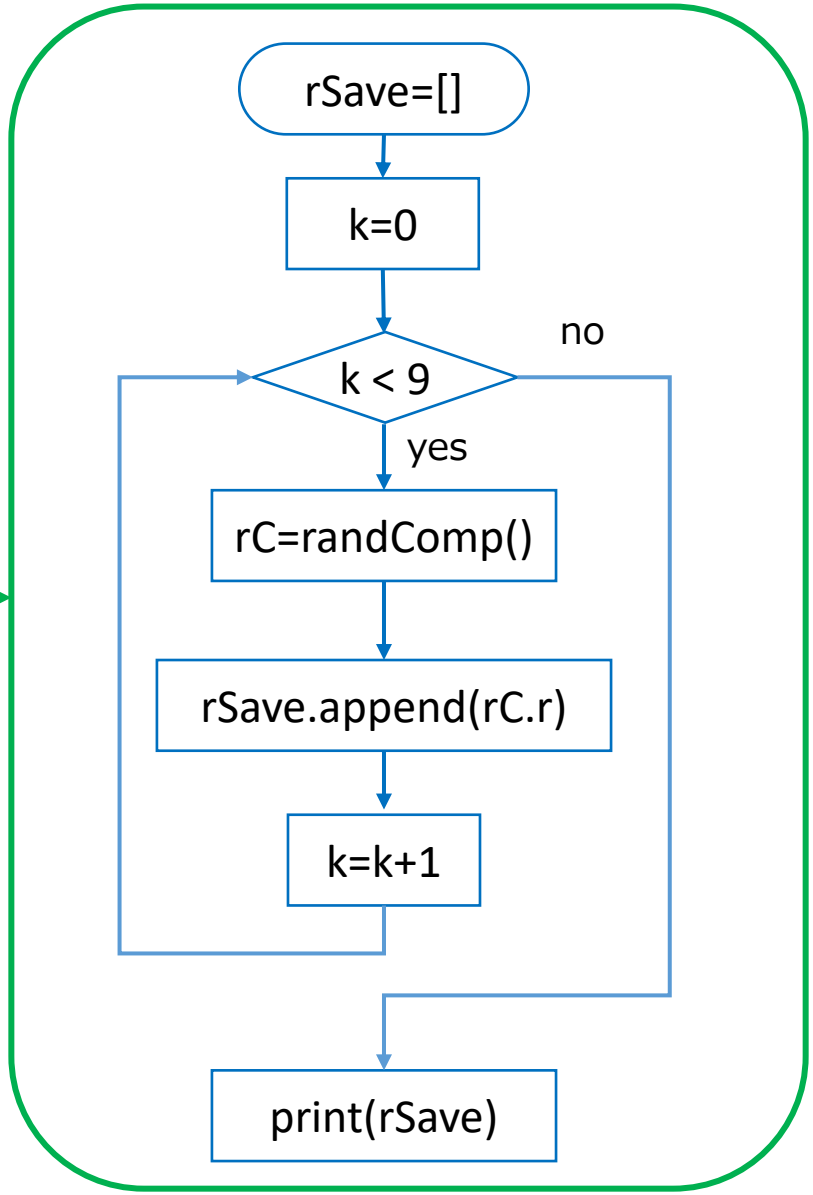
反復（フローチャート）

test.py × testFunc.py testClassClass.py testClassClassEx2.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rSave = []
20
21 for k in range(9):
22     rC = randComp()
23     rSave.append(rC.r)
24
25 print(rSave)
```

Shell

```
>>> %Run -c $EDITOR_CONTENT
  [9, 4, 8, 9, 4, 7, 2, 1, 2]
>>>
```



※ 動画説明からフローチャートを訂正

リスト

角括弧で囲みカンマで区切る

数字や文字などのデータn個を順序づけて格納

リストの構文

リスト = [データ1, データ2, …, データn]

```
test.py × | testFunc.py | testClassClass.py | testClassClassEx2.py

 1  # 教員研修用教材 123頁 をクラスで表現
 2  import random
 3
 4  class randComp:
 5      def __init__(self):
 6          self.a = 5
 7          self.n = 10
 8          self.r = random.randrange(self.n)
 9      def arComp(self):
10          if self.a == self.r:
11              print("当たり")
12          elif self.a > self.r:
13              print(str(self.a)+"の方が大きい")
14          elif self.a < self.r:
15              print(str(self.a)+"の方が小さい")
16      def __del__(self):
17          pass
18
19  rSave = []          rSaveを空のリストとして生成
20
21  for k in range(9):
22      rC = randComp()
23      rSave.append(rC.r)      For文の中でリスト型のメソッド append() を用いてrSaveに要素を追加
24
25  print(rSave)        最終的なrSaveを画面出力

Shell

>>> %Run -c $EDITOR_CONTENT
  [9, 4, 8, 9, 4, 7, 2, 1, 2]      プログラム25行目の実行結果
>>>
```

- リストはリスト型のオブジェクトです。リスト型はPythonに元から組み込まれた型であり，append() や clear() など様々なメソッドを内包しています。

リストがリスト型のオブジェクトであると意識するとメソッド append() を呼び出すことの意味を見通しよく理解できます

一見すると数字が9個並んでいるだけに見えますが，リスト型に定義されたメソッドを利用可能であり，単なる数字の羅列ではないことに注意してください

range関数の代替

test.py × testFunc.py testClassClass.py testClassClassEx2.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rSave = []
20
21 for k in range(9):
22     rC = randComp()
23     rSave.append(rC.r)
24
25 print(rSave)
```

シーケンスをrange型オブジェクトで表現

Shell

```
>>> %Run -c $EDIT
  [9, 4, 8, 9, 4, 7, 2,
>>>
```

range関数は少し分かりづらいのですが，リストやタプルなど異なるオブジェクトと置き換えても同じ働きをすることと対比すれば，range関数は複数のデータを順番に並べていると理解しやすいと思います

反復の基本構文

range関数は代替可能です

for 変数 in **シーケンス**:
    実行部

リスト：角括弧囲み，書換え可能

シーケンスをリスト型オブジェクトで表現

```
for k in [0,1,2,3,4,5,6,8]:
    rC = randComp()
    rSave.append(rC.r)
```

シーケンスをタプル型オブジェクトで表現

```
for k in (0,1,2,3,4,5,6,8):
    rC = randComp()
    rSave.append(rC.r)
```

タプル：丸括弧囲み，書換え不可能

※ 動画説明からリストとタプルを訂正
0から9まで記載していましたが、0から８まで記載するのが正しいです

二重反復

testClassClassEx3.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rSave = []
20
21 for k in range(9):
22     rC = randComp()
23     for m in range(5):
24         rSave.append(rC.r)
25
26 print(rSave)
```

for文を入れ子にすることにより二重反復が実現されます。
各kに対して、mを5回繰り返していることがポイントです。

Shell

```
>>> %Run testClassClassEx3.py
[4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 1, 1, 1, 1, 1, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 9, 9, 9, 9, 9,
8, 8, 8, 8, 8, 0, 0, 0, 0, 0, 7, 7, 7, 7, 7]
>>>
```

k=0に対して
m=0,1,2,3,4を繰返し

k=3に対して
m=0,1,2,3,4を繰返し

二重反復の基本構文

```
for 変数A in シーケンスA:
    実行部A
    for 変数B in シーケンスB:
        実行部B
```

実行部B：変数Bの実行部
for 変数B in シーケンスB: ～ 実行部B：変数Aの実行部

反復の基本構文において実行部がどのように規定されるか意識すると，二重反復を理解しやすいです

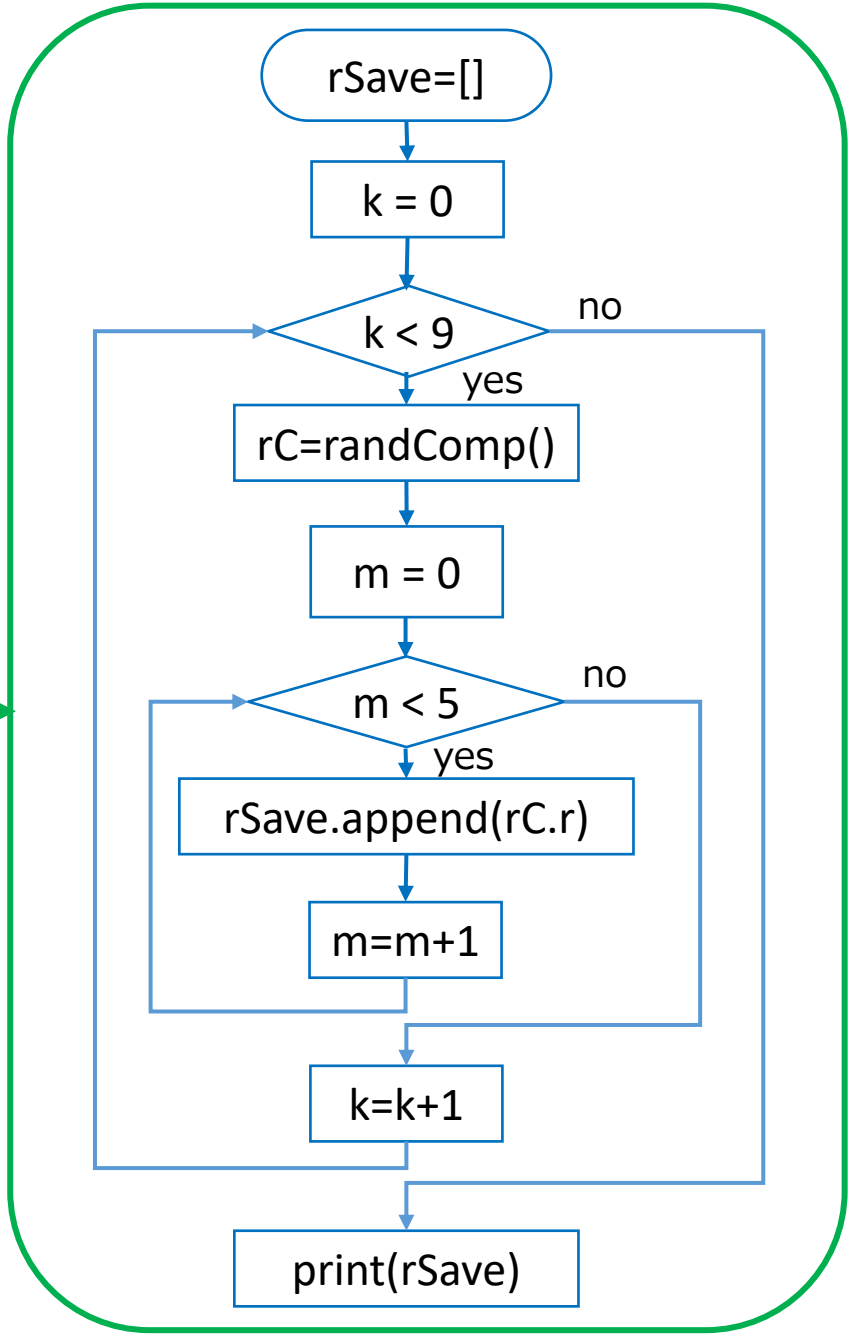
二重反復

testClassClassEx3.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rSave = []
20
21 for k in range(9):
22     rC = randComp()
23     for m in range(5):
24         rSave.append(rC.r)
25
26 print(rSave)
```

Shell

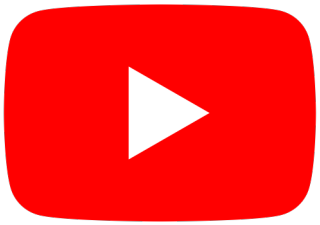
```
>>> %Run testClassClassEx3.py

  [4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 1, 1, 1, 1, 1, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 9, 9, 9, 9, 9,
  8, 8, 8, 8, 8, 0, 0, 0, 0, 0, 7, 7, 7, 7, 7]

>>>
```



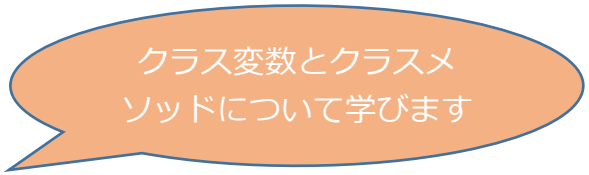
※ 動画説明からフローチャートを訂正

概要

21:11



- Python環境整備と動作確認
  - Pythonの基本１
- **オブジェクトの理解**
  - 概念
  - 例示
  - Pythonの基本２
  - **クラスの一般的表記**
- 外部パッケージの活用
  - モジュール
  - パッケージ
  - ドットの謎
- まとめ



メンバ変数の分類：

インスタンスを生成する時に定まるメンバ変数を**インスタンス変数**と呼びます．インスタンス変数はインスタンス毎に独立に定義されます．あるインスタンス変数の書き換えは別なインスタンス変数に波及しません．

クラスを定義するときに定まるメンバ変数を**クラス変数**と呼びます．異なるインスタンスに共通し，クラス変数の書き換えはすべてのインスタンスに波及します．

クラス

メンバ変数

インスタンス変数
- 車体カラーを選択可能
- カーナビを設置可能
- アクセルの傾きは0度から10度の範囲内
- ハンドルの傾きは±540度の範囲内

クラス変数
- 車長は1800mmに設定
- 排気量は1000mlに設定

オブジェクト (インスタンス)

ID: 4289…

インスタンス変数
- 製造時に車体カラーは赤を選択 ← 文字列型
- 製造時にカーナビ**を設置 ← オブジェクト
- 出荷時のアクセルの傾きは0度 ← 浮動小数点型
- 出荷時のハンドルの傾きは0度 ← 浮動小数点型

クラス変数
- 車長は1800mm ← 整数型
- 排気量1000mlのエンジンを搭載 ← オブジェクト

メンバ関数の分類：

インスタンスを生成する際に定まるメンバ関数を**インスタンスメソッド**と呼びます．インスタンスに付随して呼び出します．主としてインスタンス変数に依存します．

クラスを生成する際に定まるメンバ関数を**クラスメソッド**と呼びます．インスタンスとは無関係に呼び出すことができます．主としてクラス変数に依存します．

スタティックメソッドと呼ばれるメンバ関数もありますが省略します．

クラス

オブジェクト (インスタンス)

ID: 4289…

メンバ関数 (メソッド)

インスタンスメソッド
- アクセル踏むことができる
- ブレーキを踏むことができる
- ハンドルを回すことができる

クラスメソッド
- 発煙筒で煙を出せる
- 工具を使える

インスタンスメソッド
- アクセルを踏む走る
- ブレーキを踏んだら止まる
- ハンドルを回すことができる

← インスタンス化したら使用可能．
インスタンス変数に主として依存．

クラスメソッド
- 発煙筒で煙を出せる
- 工具を使える

← インスタンス化の有無に関わらず使用可能．
クラス変数に主として依存．

サンプルプログラム（クラスの書式を例示）

test.py | testFunc.py | testClassClass.py | testClassInstance.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp3:
5      a = 5
6      n = 10
7      r = random.randrange(n)
8      @classmethod
9      def arComp3(cls):
10         if cls.a == cls.r:
11             print("当たり")
12         elif cls.a > cls.r:
13             print(str(cls.a)+"の方が大きい")
14         elif cls.a < cls.r:
15             print(str(cls.a)+"の方が小さい")
16
17 randComp3.arComp3()
18
```

Shell

```
>>> %Run testClassInstance.py
  5の方が大きい
>>>
```

クラス変数とクラスメソッドを含む場合の書式：

```
class クラス名:
    クラス変数
    def __init__(self, 引数):
        コンストラクタの実行部
    def インスタンスメソッド(self, 引数):
        メソッドの実行部
    @classmethod
    def クラスメソッド(cls, 引数):
        クラスメソッドの実行部
    def __del(self):
        デストラクタの実行部
```

補足：

クラス変数とインスタンス変数，クラスメソッドとインスタンスメソッドは共存可能．左のサンプルではクラス変数とインスタンス変数のみを含む書き方．

サンプルプログラム（クラスメソッドを例示）

test.py | testFunc.py | testClassClass.py | testClassInstance.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp3:
5      a = 5
6      n = 10
7      r = random.randrange(n)
8      @classmethod
9      def arComp3(cls):
10         if cls.a == cls.r:
11             print("当たり")
12         elif cls.a > cls.r:
13             print(str(cls.a)+"の方が大きい")
14         elif cls.a < cls.r:
15             print(str(cls.a)+"の方が小さい")
16
17 randComp3.arComp3()
18
```

Shell

```
>>> %Run testClassInstance.py
  5の方が大きい
>>>
```

今度はクラス変数とクラスメソッドを例示する．

5行目から7行目にかけてクラス変数 a, n, r を定義している．

コンストラクタを経ずにいきなり変数が定義されていることに注意されたい．

8行目から15行目にかけてクラスメソッド arComp3() を定義している．変数 cls はクラス自身を意味する．クラス変数 cls.a や cls.r はクラス内部で利用できる．

クラス外部からクラス変数にアクセスするには，randComp3.a や randComp3.r などと表記する．

17行目においてクラスメソッド arComp3() を呼び出し，教員研修用教材123ページのサンプルプログラムと同等の実行結果が得られた．

インスタンスを生成することなくクラス外部でクラスメソッドを呼び出せたことを確認できた．

サンプルプログラム（クラスメソッドを例示）

test.py × testFunc.py testClassClass.py testClassInstance.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp4:
5      a = 5
6      n = 10
7      r = random.randrange(n)
8      @classmethod
9      def arComp4(cls,x,y):
10         if x == y:
11             print("当たり")
12         elif x > y:
13             print(str(x)+"の方が大きい")
14         elif x < y:
15             print(str(x)+"の方が小さい")
16
17 randComp4.arComp4(randComp4.a,randComp4.r)
18
19 randComp4.arComp4(4,8)
20
```

Shell

```
>>> %Run testClassInstance.py

  当たり
  4の方が小さい

>>>
```

クラスメソッドをクラス外で単独利用可能とするための設計変更例も例示する.

クラスメソッド arComp3() ではクラス変数を使用していたが，クラスメソッド arComp4() ではクラス変数を使用せずに引数 x, y を取る形で設計変更した.

17行目はクラスメソッド arComp4() にクラス変数 randComp4.a と randComp4.r を渡しており，教員研修用教材123ページのサンプルプログラムと同等の実行結果が得られた.

さらに19行目ではクラスメソッド arComp4() に引数 4 と 8 を渡しており，クラス変数とは無関係にクラスメソッド arComp4() を単体で使用可能であることを確認できる.

サンプルプログラム（デフォルト引数を例示）

testClassInstance5.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp5:
5      a = 5
6      n = 10
7      r = random.randrange(n)
8      @classmethod
9      def arComp5(cls, x=None, y=None):
10         if x == None:
11             x = cls.a
12         if y == None:
13             y = cls.r
14         if x == y:
15             print("当たり")
16         elif x > y:
17             print(str(x)+"の方が大きい")
18         elif x < y:
19             print(str(x)+"の方が小さい")
20
21 randComp5.arComp5() # デフォルト引数を適用
22 randComp5.arComp5(4,8) #  引数差し替え
23 randComp5.arComp5(x=4,y=8) #  引数差し替え
24
```

Shell

```
>>> %Run testClassInstance5.py

  5の方が小さい
  4の方が小さい
  4の方が小さい

>>>
```

デフォルト引数という考え方についても紹介する．

デフォルト引数とは関数定義時の仮引数を意味する．関数呼び出し時に引数を指定して仮引数から差し替えることも可能だし，引数を指定せずに仮引数をそのまま引数とすることも可能である．

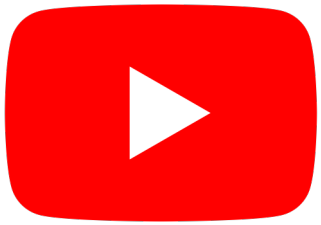
9行目ではデフォルト引数として None（空白を意味する特殊なオブジェクト）を指定している．10行目から13行目にかけて，関数呼び出し時に引数を指定しない場合は仮引数をクラス変数で差し替えることにより引数を確定する処理を行うことが述べられている．

21行目では引数を指定せずにデフォルト引数を適用した例が示されており，22行目と23行目では引数を指定した例が示されている．

クラス randComp4 の例と比較すると，デフォルト引数を用いることによりクラスメソッドの使い勝手が改善されていることがわかる．

概要

22:33



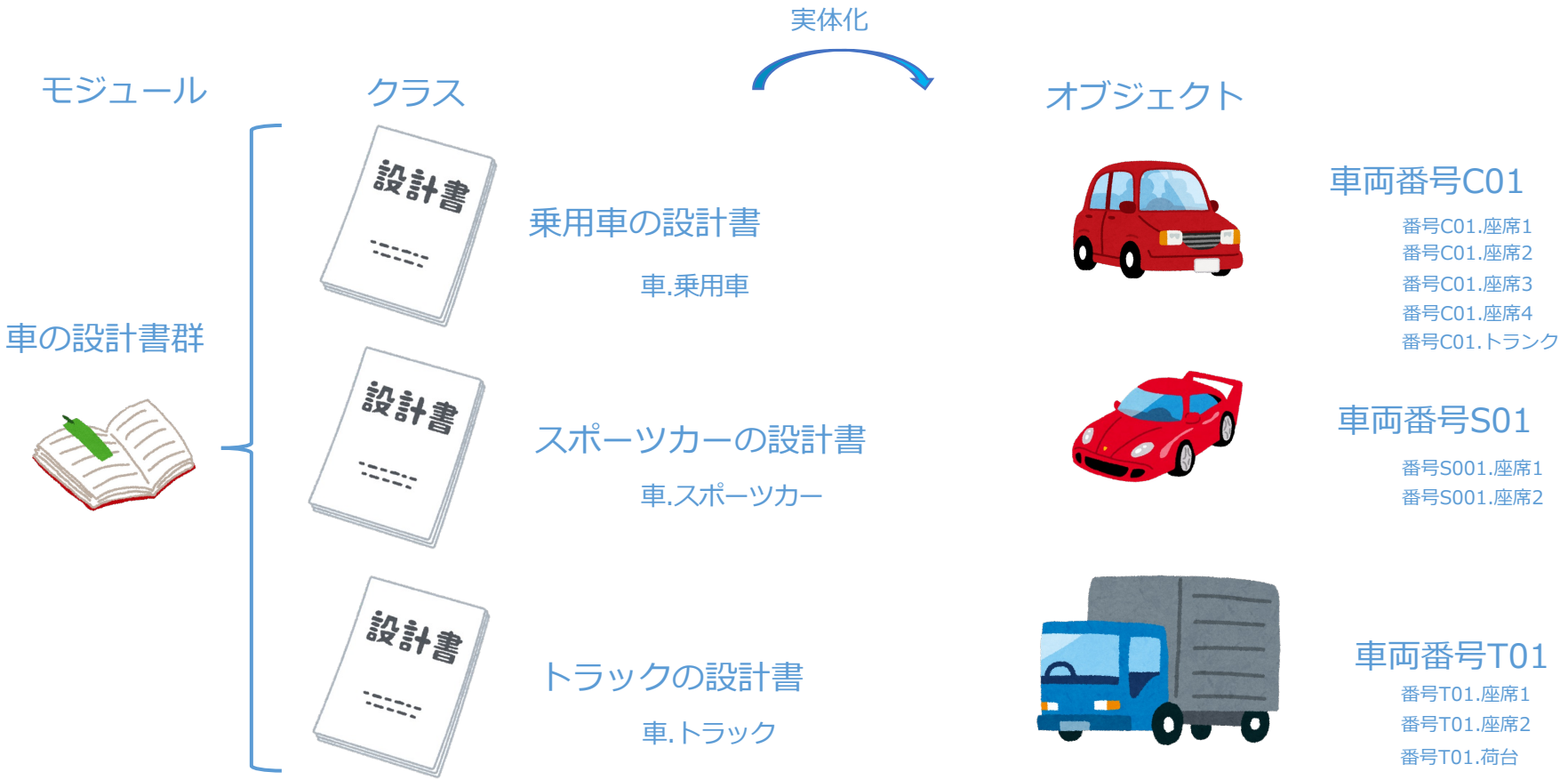
- Python環境整備と動作確認
  - Pythonの基本１
- オブジェクトの理解
  - 概念
  - 例示
  - Pythonの基本２
  - クラスの一般的表記
- **外部パッケージの活用**
  - **モジュール**
  - パッケージ
  - ドットの謎
- まとめ

便利な道具を再活用

※ Google Colaboratory →プログラム作成をスキップして説明内容をご確認下さい。
Google drive をマウントが必要となり若干難しくなります。

モジュール

モジュール：変数・関数・クラスの束（Pythonプログラム）



Pythonには標準モジュールが多数内包されているが，外部モジュールも充実しています．自作モジュールを作成して利用することは可能ですが，様々な機能を実現するために適切な外部モジュールを探して使用することが欠かせません．

サンプルプログラム（クラス例示を再掲載）

test.py　testFunc.py　testClass.py

```
1  # 教員研修用教材 123頁 を関数で表現
2  import random
3
4  class randComp:
5      def __init__(self):
6          self.a = 5
7          self.n = 10
8          self.r = random.randrange(self.n)
9      def arComp(self):
10         if self.a == self.r:
11             print("当たり")
12         elif self.a > self.r:
13             print(str(self.a)+"の方が大きい")
14         elif self.a < self.r:
15             print(str(self.a)+"の方が小さい")
16     def __del__(self):
17         pass
18
19 rC = randComp()
20 rC.arComp()
21
```

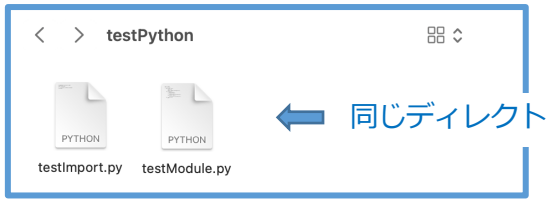
Shell

```
>>> %Run testClass.py
  5の方が大きい
>>>
```

課題2-2で例示したサンプルプログラムに対して，クラス randComp のモジュール化を例示する.

2行目から17行までをモジュール (独立したPythonファイル) に分割し，モジュールを読み込んでから19行目と20行目を実行する形となるようファイル構成を変更する.

課題 3-1:



test.py × | testFunc.py | testClassClass.py | testModule.py | testImport.py

```
1  import random
2
3  class randComp:
4      def __init__(self):
5          self.a = 5
6          self.n = 10
7          self.r = random.randrange(self.n)
8      def arComp(self):
9          if self.a == self.r:
10             print("当たり")
11         elif self.a > self.r:
12             print(str(self.a)+"の方が大きい")
13         elif self.a < self.r:
14             print(str(self.a)+"の方が小さい")
15     def __del__(self):
16         pass
17
```

testClassClass.py × | testClassInstance.py | testImport.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import testModule
3
4  rC = testModule.randComp()
5  rC.arComp()
6
```

Shell

```
>>> %Run testImport.py
  5の方が小さい
```

課題 3-1:

課題 2-2 のサンプルプログラム testClass.py を分割します.

1) ファイル testModule.py と testImport.py を作成し，共通のディレクトリ内部に配置します.

2) サンプルプログラム testClass.py の2行目から17行目をファイル testModule.py に転記して下さい.

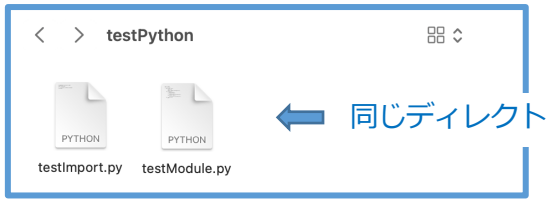
3) ファイル testImport.py に画面左下のスクリプトを転記して下さい．コンストラクタの記載方法がサンプルプログラム testClass.py と若干異なることに注意して下さい.

4) ファイル testimport.py を実行して下さい.

実行結果がサンプルプログラム testClass.py の実行結果と変わらぬことを確認して下さい.

※ Google Colaboratory → プログラム作成をスキップして説明内容をご確認下さい。

サンプルプログラム（モジュール作成例）



test.py | testFunc.py | testClassClass.py | testModule.py | testImport.py

```
1  import random
2
3  class randComp:
4      def __init__(self):
5          self.a = 5
6          self.n = 10
7          self.r = random.randrange(self.n)
8      def arComp(self):
9          if self.a == self.r:
10             print("当たり")
11         elif self.a > self.r:
12             print(str(self.a)+"の方が大きい")
13         elif self.a < self.r:
14             print(str(self.a)+"の方が小さい")
15     def __del__(self):
16         pass
17
```

testClassClass.py | testClassInstance.py | testImport.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import testModule
3
4  rC = testModule.randComp()
5  rC.arComp()
6
```

Shell

```
>>> %Run testImport.py
  5の方が小さい
```

ファイル testModule.py にはクラス randComp が含まれており，モジュールを作成できたことがわかります．ここでは単一のクラスしか含まれていませんが，モジュールに複数の関数やクラスを含めることが可能です．

ファイル testImport.py を用いた自作モジュールの使用例を解読します．

1行目に自作モジュール testModule を読み込む．4行目にて testModule.randComp() を実行しインスタンス rC を作成する．5行目にてインスタンスメソッド rC.arComp() を呼び出し，サンプルプログラム testClass.pyと同等の実行結果が得られた．

ドット「.」によりモジュールとクラスの間の**階層構造**を表す．例えば，「testModule.randComp()」は「モジュールtestModuleのクラスrandCompの初期化」を意味する．

サンプルプログラム（モジュール作成例２）

test.py × | testFunc.py | testClassClass.py | testModule.py | testImport.py

```
1  import random
2
3  class randComp:
4      def __init__(self):
5          self.a = 5
6          self.n = 10
7          self.r = random.randrange(self.n)
8      def arComp(self):
9          if self.a == self.r:
10             print("当たり")
11         elif self.a > self.r:
12             print(str(self.a)+"の方が大きい")
13         elif self.a < self.r:
14             print(str(self.a)+"の方が小さい")
15     def __del__(self):
16         pass
17
```

testClassClass.py × | testModule.py | testImport.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  from testModule import randComp
3
4  rC = randComp()
5  rC.arComp()
6
```

Shell

```
>>> %Run testImport.py
  5の方が小さい
```

モジュール testModule.py は前頁と共通として，前頁とは異なるモジュールの読み込み方を例示する．

ファイル testImport.py の1行目に testModule からクラス randComp を指定して読み込む．

4行目では randComp() を実行しインスタンスメソッド rC を作成する．

以降は前頁と同様である．

補足：

このサンプルプログラムではクラス randComp 直接的に読み込んでいるのでモジュール testModule との階層構造を表す必要はない．逆に，4行目において randComp() の代わりに testModule.randComp() として呼び出すとエラーが発生する．

課題 3-2:

test.py | testFunc.py | testClassClass.py | testClassInstance.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import random
3
4  class randComp3:
5      a = 5
6      n = 10
7      r = random.randrange(n)
8      @classmethod
9      def arComp3(cls):
10         if cls.a == cls.r:
11             print("当たり")
12         elif cls.a > cls.r:
13             print(str(cls.a)+"の方が大きい")
14         elif cls.a < cls.r:
15             print(str(cls.a)+"の方が小さい")
16
17 randComp3.arComp3()
18
```

Shell

```
>>> %Run testClassInstance.py
  5の方が大きい
>>>
```

課題 3-2:

クラスメソッドを例示にて提示した randComp3 を testModule3.py としてモジュール化して下さい.

さらに，モジュール testModule3 を読み込み，クラスメソッド arComp3() を実行するプログラムを作成して下さい.

補足：

クラスの説明の難易度が高く感じた場合はスキップして頂いて構いません.

※ Google Colaboratory → プログラム作成をスキップして説明内容をご確認下さい。

回答 3-2:

testImport.py　testModule3.py

```
1  import random
2
3  class randComp3:
4      a = 5
5      n = 10
6      r = random.randrange(n)
7      @classmethod
8      def arComp3(cls):
9          if cls.a == cls.r:
10             print("当たり")
11         elif cls.a > cls.r:
12             print(str(cls.a)+"の方が大きい")
13         elif cls.a < cls.r:
14             print(str(cls.a)+"の方が小さい")
15
```

testImport.py　testModule3.py

```
1  # 教員研修用教材 123頁 をクラスで表現
2  import testModule3
3
4  testModule3.randComp3.arComp3()
5
```

Shell

```
>>> %Run testImport.py
  5の方が大きい
```

課題 3-2:

クラスメソッドを例示にて提示した randComp3 を testModule3.py としてモジュール化して下さい.

さらに，モジュールを読み込み，クラスメソッド arComp3() を実行するプログラムを作成して下さい.

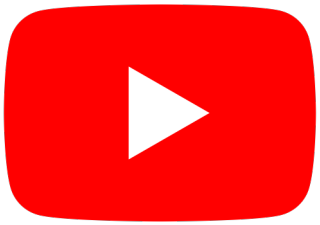
回答 3-2:

モジュールの作成方法は前頁と同様であり，外部モジュール random と自作クラス randComp3 を自作モジュール testModule3.py に保存する.

クラスメソッド arComp3() を呼び出す際は，ドット「.」を重ねて階層構造を指定する．一つ目のドットはモジュールとクラスの階層構造を表し，二つ目のドットはクラスとメソッドの階層構造を表す.

概要

17:57

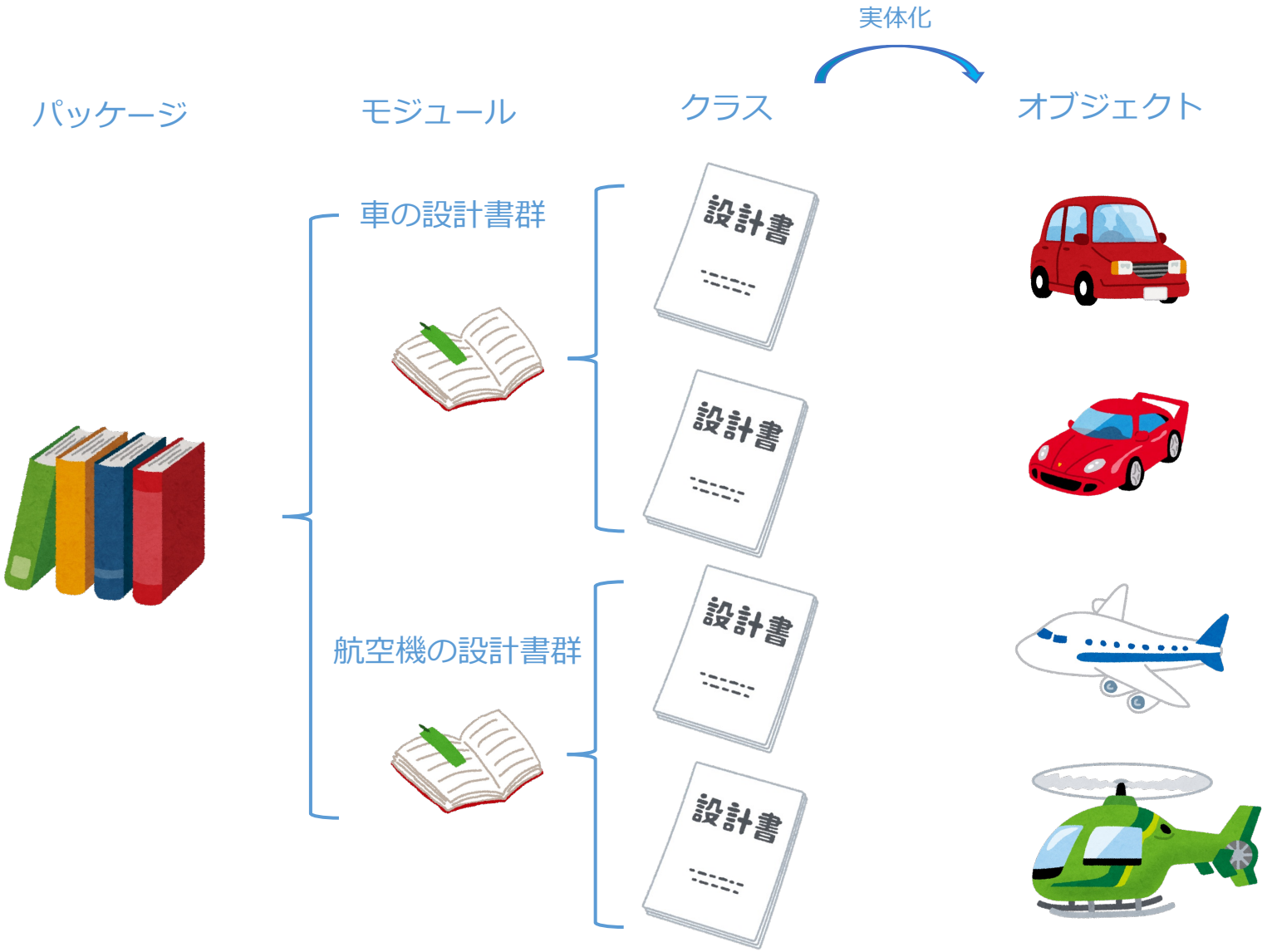


- Python環境整備と動作確認
  - Pythonの基本１
- オブジェクトの理解
  - 概念
  - 例示
  - Pythonの基本２
  - クラスの一般的表記
- **外部パッケージの活用**
  - モジュール
  - **パッケージ**
  - ドットの謎
- まとめ

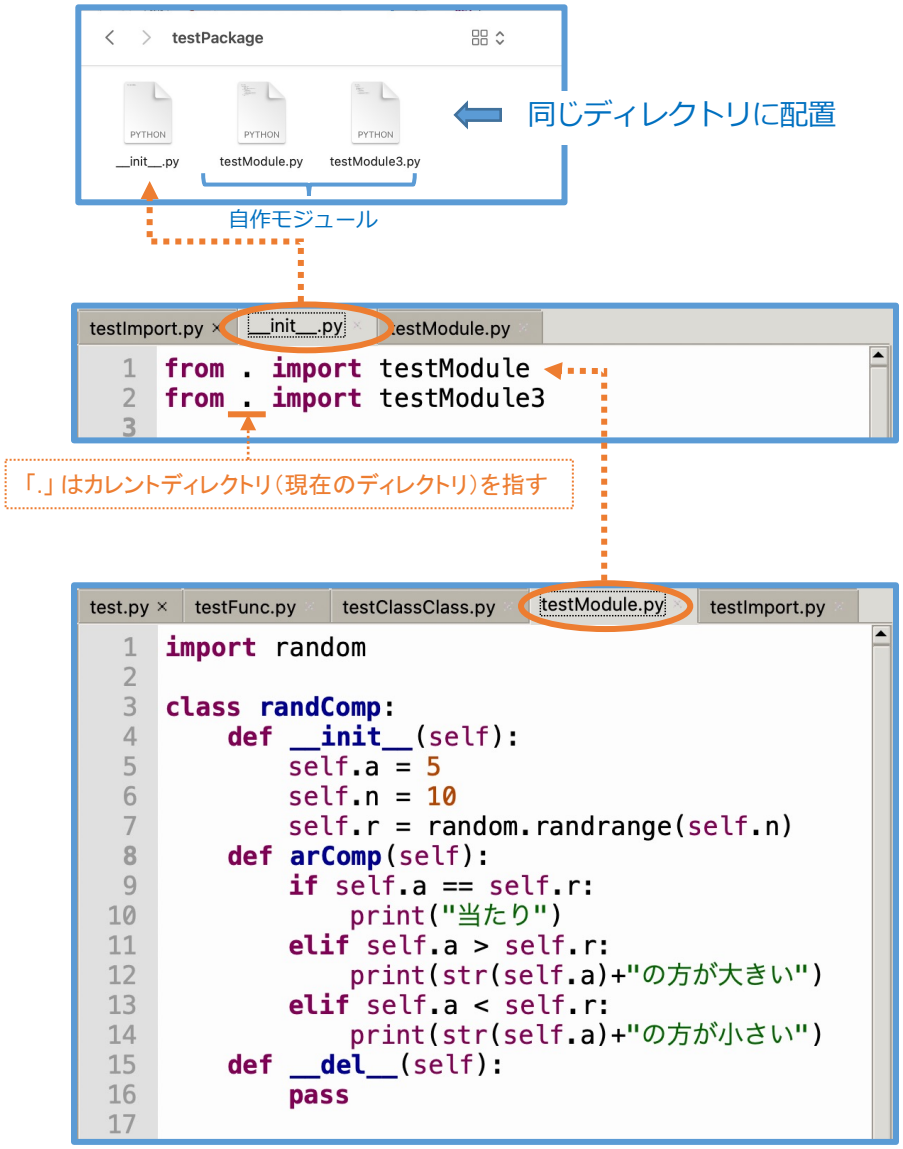
※ Google Colaboratory →プログラム作成をスキップして説明内容をご確認下さい。
Google drive をマウントが必要となり若干難しくなります。

パッケージ

パッケージ：モジュールの束（Pythonプログラムの束）



サンプルプログラム（自作パッケージの作成）



自作パッケージの作成について例示する．

ディレクトリ testPackage の中にファイル __init__.py と自作モジュールを配置します．

ファイル __init__.py はディレクトリがパッケージであることを示すための特別なファイルです．

import文における「.」はファイル __init__.py が存在するカレントディレクトリを表します．

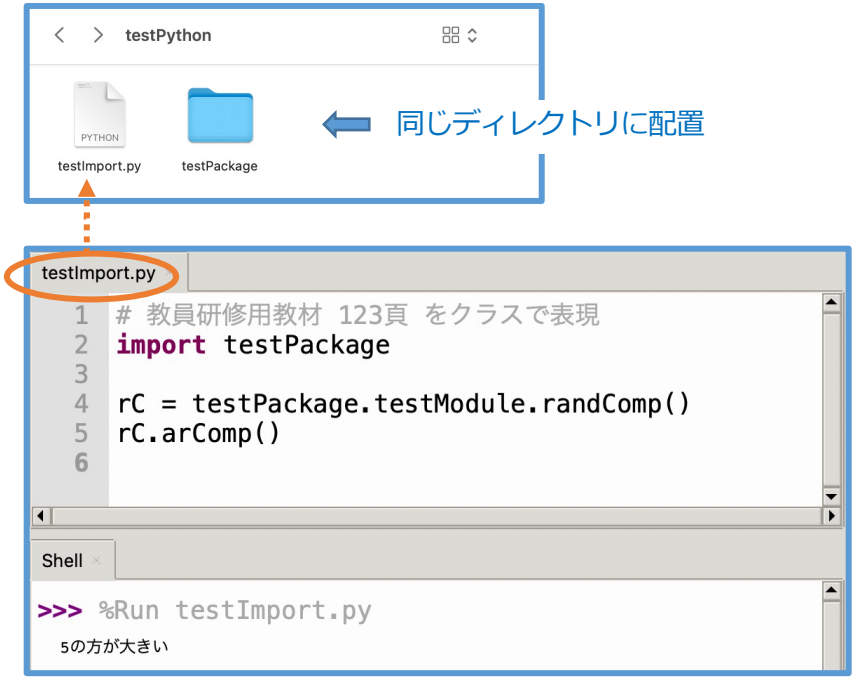
1行目と2行目は現在のディレクトリからモジュール testModule と testModule3 を読込むことを宣言しています．

以上により，自作パッケージ testPackage が導入されます．

なお，本講義資料ではモジュール testModule3.py を提示していません．本例題の動作確認を行う場合は testModule.py のコピーなど適当に作成して下さい．

※ Google Colaboratory → プログラム作成をスキップして説明内容をご確認下さい。

サンプルプログラム（自作パッケージの読込）



自作パッケージの使用方法を例示する.

ディレクトリtestPackage と同じディレクトリに Pythonプログラム testImport.py を作成します.

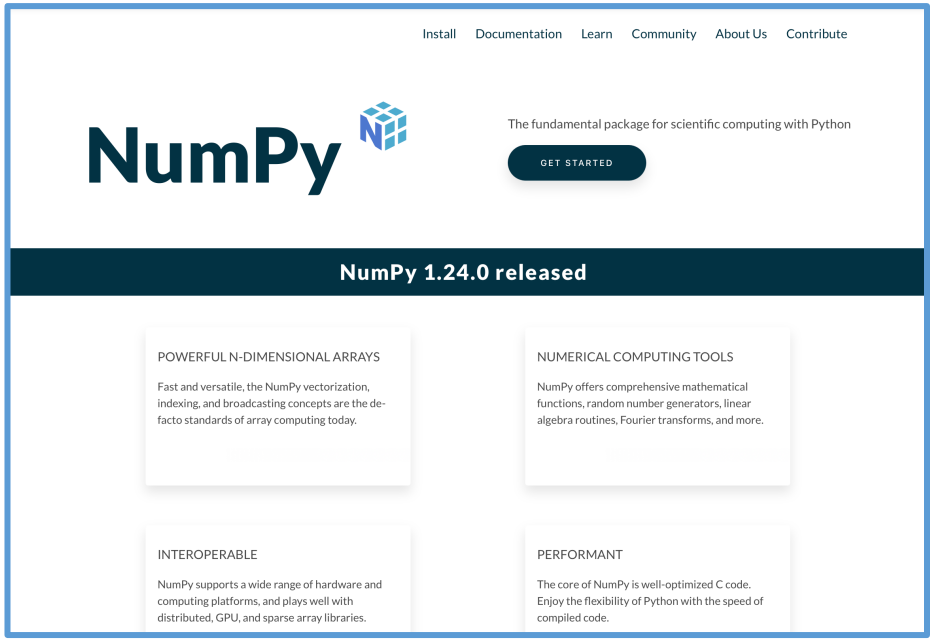
2行目においてimport文により自作パッケージを読み込む.

4行目において，自作パッケージ testPackage の下の自作モジュール testModule の中のクラス randComp を適用してインスタンス rC を生成するという形で記載する.

ドット「.」により階層構造を表す．一つ目のドットはパッケージとモジュールの間の階層構造を表し，二つ目のドットはモジュールとクラスの間の階層構造を表す.

5行目におけるインスタンスメソッド rC.arComp() の呼出しはこれまでの例題と同様で，教員研修用教材123ページのサンプルプログラムと同等の実行結果が得られた.

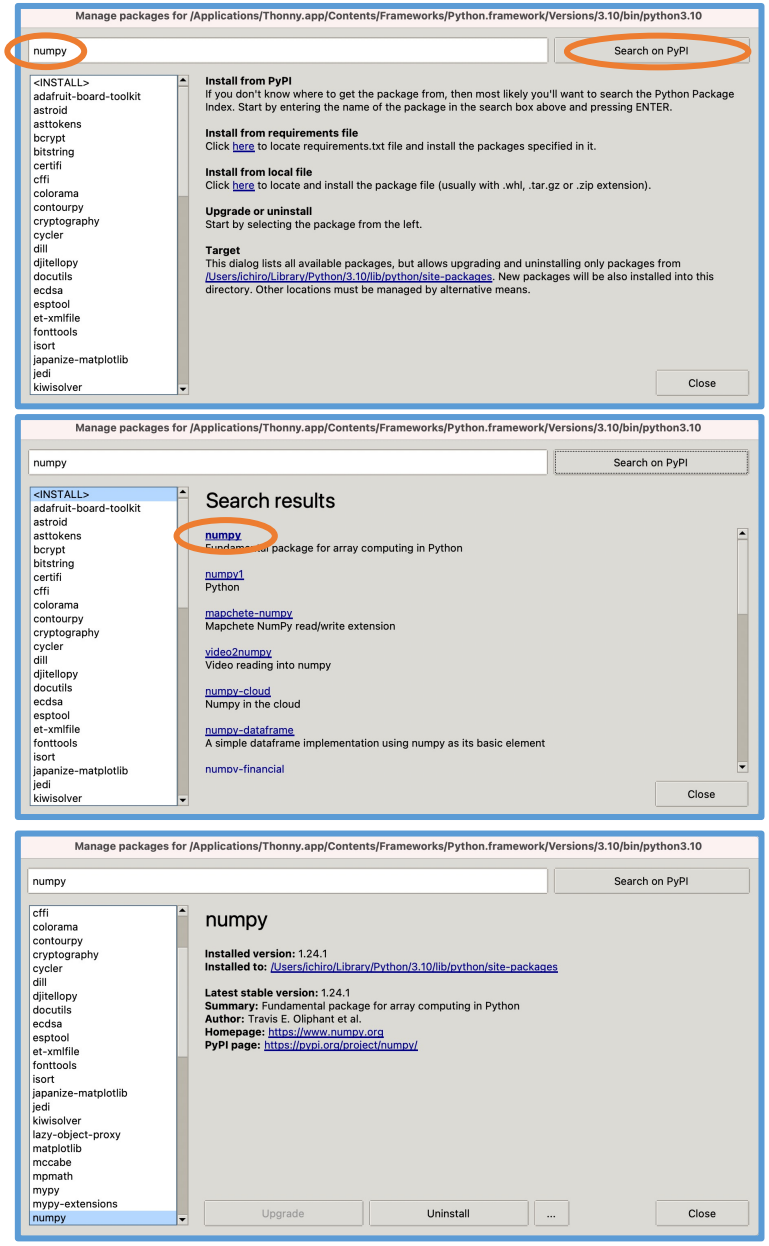
外部パッケージ



https://numpy.org

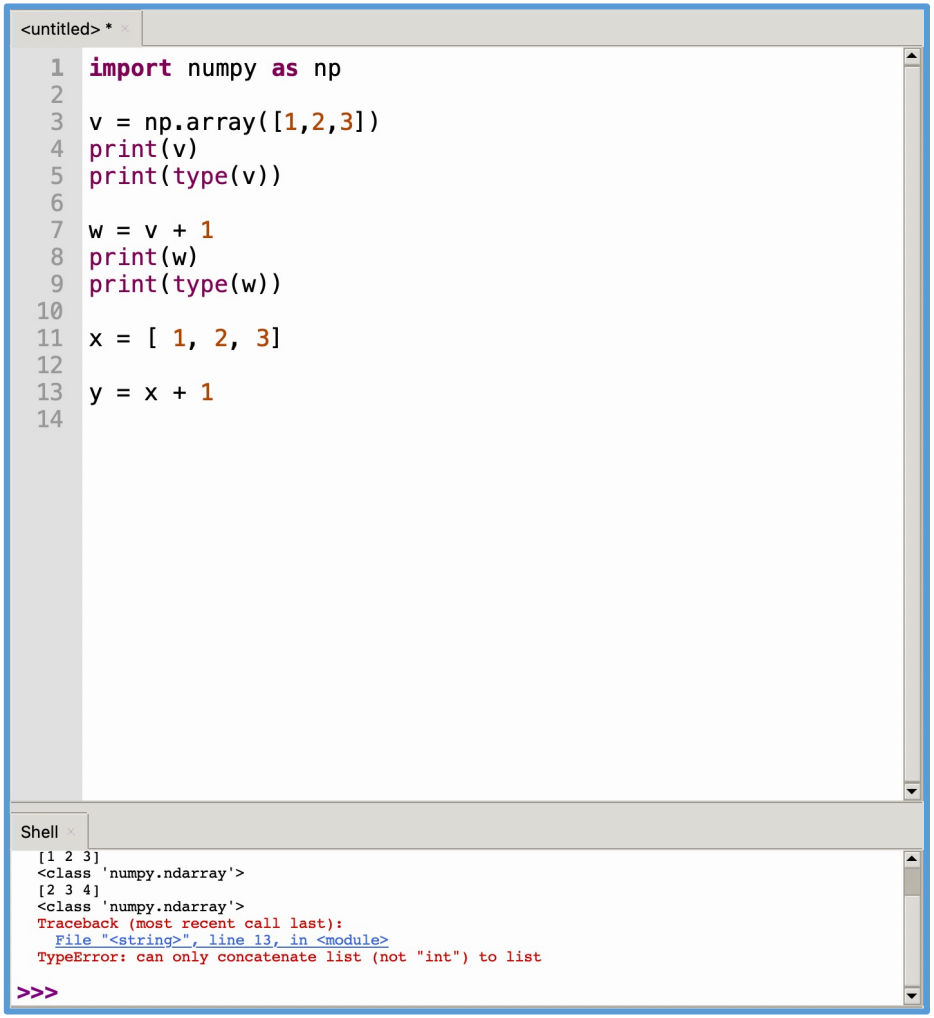
- Pythonの機能拡張を行おうとすると外部パッケージの追加が必須.
  - 例えば、NumPy は行列計算をするために有名なパッケージです. 画像処理や機械学習など様々な外部パッケージでもバックグラウンドで使用される.
  - 他に，作図するなら matplotlib, csvファイルの入出力をするなら pandas，画像処理をするなら cv2など，定番の外部パッケージが多々知られる.

外部パッケージの追加方法



- Thonnyから外部パッケージを追加するなら，メニューの「Tools」から「Manage Packages」を開く．パッケージ検索画面が現れます．
- 続けて，検索メニューからパッケージを検索．
- Pythonから参照可能なディレクトリ内にパッケージがインストールされる．
- Pythonプログラムからパッケージを import することにより外部パッケージを読み込むことができる．

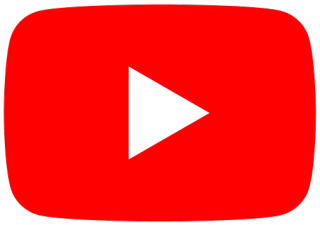
外部パッケージの使用例



- 左の例では NumPy の使用例を示す.
- 1行目では，外部パッケージ NumPy を np という簡略名称で読み込む.
- 3行目では，numpy.ndarray型のオブジェクト v を作成します．4行目にて v の成分が [1 2 3] であることが確認される.
- 一見 v はリストと類似しているが，全然異なる機能を備えていることに注意されたい.
- 例えば，7行目を見ると v に 1 を加えると全要素に 1 が足されることがわかる．11行目から13行目を見るとリストでは同様の演算が成立しない．numpy.ndarray型とlist型は全く異なることが確認できる.

概要

6:16



- Python環境整備と動作確認
  - Pythonの基本１
- オブジェクトの理解
  - 概念
  - 例示
  - Pythonの基本２
  - クラスの一般的表記
- **外部パッケージの活用**
  - モジュール
  - パッケージ
  - **ドットの謎**
- まとめ

ドットの役割を確認

ドット「.」の謎

- ドットは正確には演算子「属性参照」と呼びます.
- Pythonには様々なドットの用法があり慣れるまでは混乱しがちですが階層構造を指定します．つまり，Pytyonには名前の衝突を避けるために「名前空間」という概念があり，名前空間の階層構造を指定しています.
- ドットの用法を判別できればプログラムの解読が容易になりますし，習熟すれば類推が効いて外部モジュールの仕様を確認しながら自分でプログラムを書く助けとなります.
- ドットの用法の組み合わせ事例
  - パッケージ.モジュール
  - モジュール.クラス
  - オブジェクト.メソッド
- 組み合わせとしてドットが重なる書き方も可能です.
  - パッケージ.モジュール.関数
  - オブジェクトA.オブジェクトB.オブジェクトBのメソッド ⇦ オブジェクトAのメンバ変数にオブジェクトBを格納してからオブジェクトBのメソッドを呼び出すという意味

教員研修用教材の「ドット」を解読

■乱数を用いたプログラムの例

ある値 a があるときに，0 ～ 9 までの数をランダムに発生させる乱数 r と比較して，大きい場合に「a の方が大きい」，小さい場合に「a の方が小さい」，等しい場合に「当たり」と表示するプログラムを図表 4 に示す。

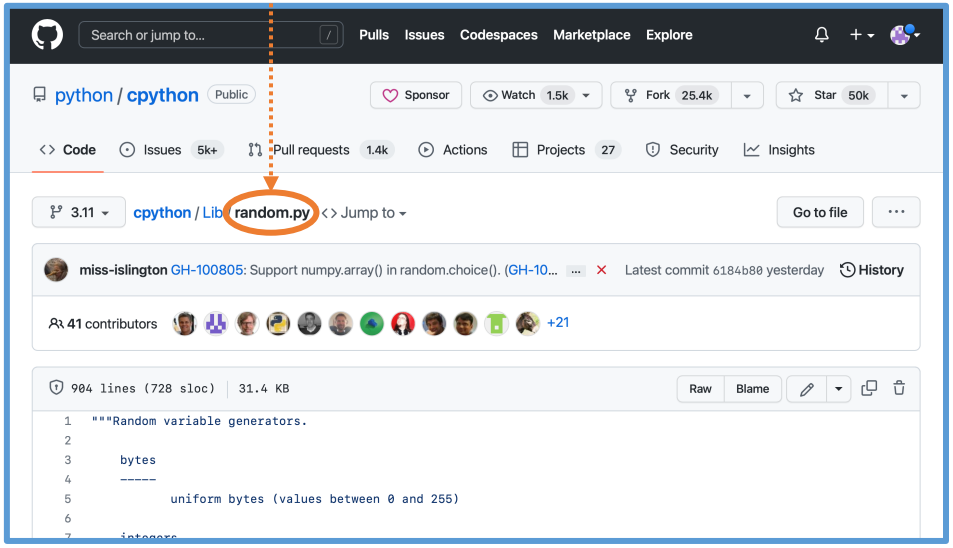
Python で乱数を扱う場合は random モジュールを import 文で読み込む必要があり，「random.randrange(10)」で 0 ～ 9 の 10 個の整数をランダムに発生させる乱数として表現している。

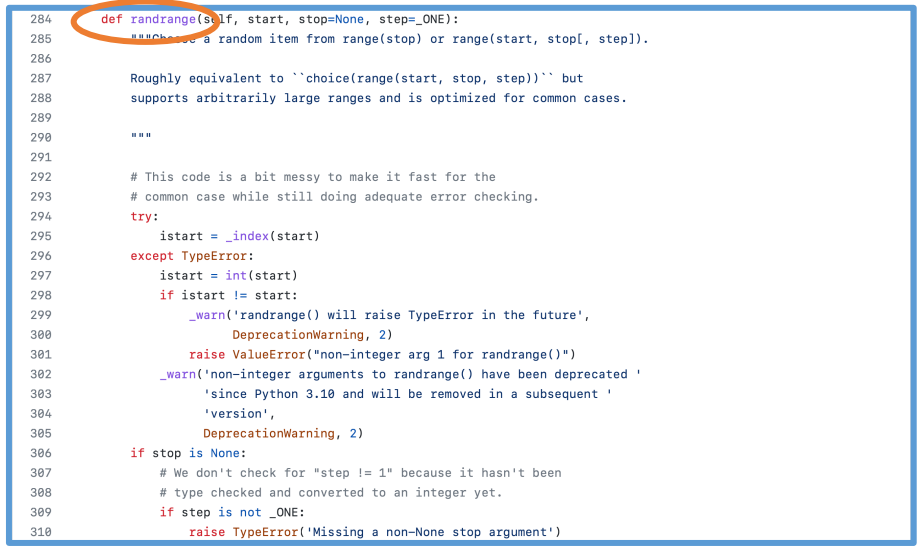
```
1  import random                #random モジュールを読み込む
2  a = 5
3  r = random.randrange(10)     #0 ～ 9 までの整数をランダムに発生
4  if a==r:
5      print(" 当たり ")
6  elif a>r:
7      print("a の方が大きい ")
8  elif a<r:
9      print("a の方が小さい ")
```

図表 4　乱数を用いたプログラムの例 1

Randomモジュールの実体は random.py というPythonファイル

Pythonファイル random.py から
関数 randrange() を呼び出していることを意味する

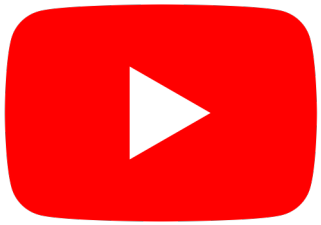




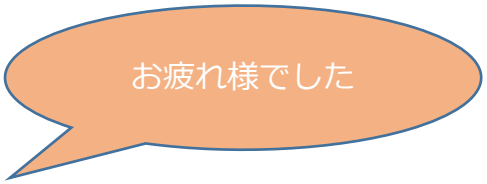
https://github.com/python/cpython/blob/3.11/Lib/random.py

概要

2:58



- Python環境整備と動作確認
  - Pythonの基本１
- オブジェクトの理解
  - 概念
  - 例示
  - Pythonの基本２
  - クラスの一般的表記
- 外部パッケージの活用
  - モジュール
  - パッケージ
  - ドットの謎
- **まとめ**



まとめ

教員研修用教材のサンプルプログラムを題材として，

- Python環境整備と動作確認
- オブジェクトの理解
- 外部パッケージの活用

に取り組んだ．

関数・クラスをうまく設計してモジュール化できればプログラムの可読性が向上してよいプログラムを書けるようになります．

課題研究等のために書籍等のサンプルプログラムを活用する場合は，ドット（属性参照）により定まる階層構造を意識し，クラスの設計（特にメソッドの設計）を確認することにより，サンプルプログラムの解読が容易となります．プログラム作成の助けとなることも期待できます．

オブジェクトはPythonに限らず様々な言語に見られる考え方であり，オブジェクトを知ることによりプログラミング言語を深く知ることができます．