青森県高等学校教育研究会商業部会
ビジネス情報分野研究委員会
令和２年度プログラミング指導法研修会

2020/08/18 – 19

＠青森大学

スケジュール

- ８月１８日（火）
  - ９：４５～　　受　付
  - １０：００～１２：００　　講義・演習１
    - Pythonを使ってできること
    - Pythonによるプログラミングの基礎知識
    - 変数とデータ型
  - １２：００～１３：００　　昼食・休憩
  - １３：００～１６：００　　講義・演習２
    - コレクション（配列の利用）
    - 条件分岐
    - 繰り返し
    - オリジナルの関数（プログラムの部品化）
- ８月１９日（水）
  - １０：００～１２：００　　講義・演習３
    - オブジェクトの概念（メソッドとプロパティ）
    - モジュールの利用
  - １２：００～１３：００　　昼食・休憩
  - １３：００～１４：５０　　講義・演習４
    - 外部ライブラリの利用
    - Pythonの活用例（データベース操作、ＧＵＩプログラムの制作等）
  - １４：５０～１５：００　　閉　会

プログラミング言語Python

特徴

リソース

開発環境

Pythonについて

1991年リリース
ABC言語の後継

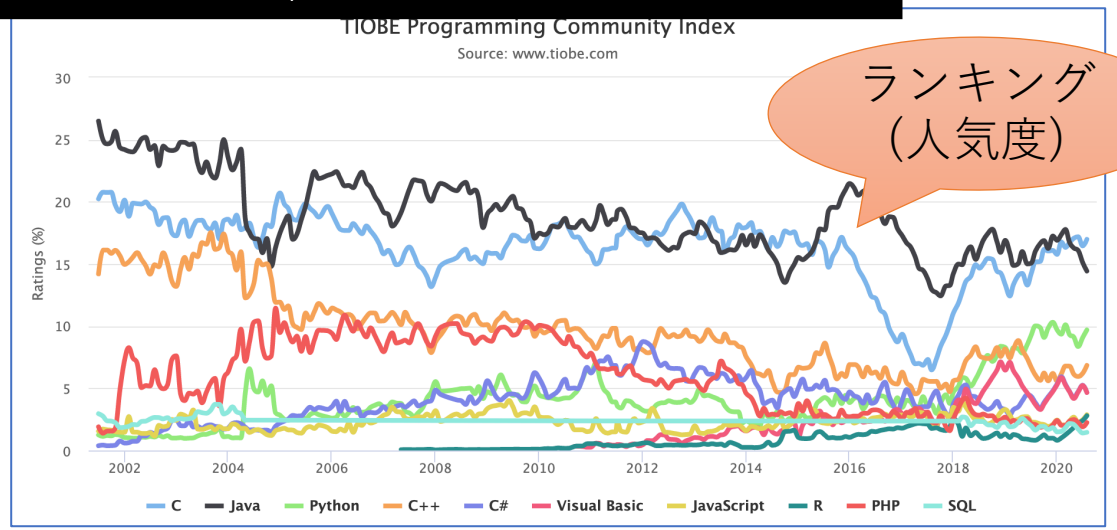
Pythonプログラマを
「蛇使い」と呼ぶことも

- Pythonの特徴
  - スクリプト言語、インタプリタ型
  - 文法がシンプルで読みやすい
  - 拡張機能(パッケージ、ライブラリ)が豊富
  - データサイエンス、AI、Webアプリ系の開発で人気
- Pythonのリソース
  - 公式ページ
    - https://www.python.org/
  - 日本のユーザグループ、ドキュメント(和訳)
    - https://www.python.jp/
  - Python Package Index
    - https://pypi.org/

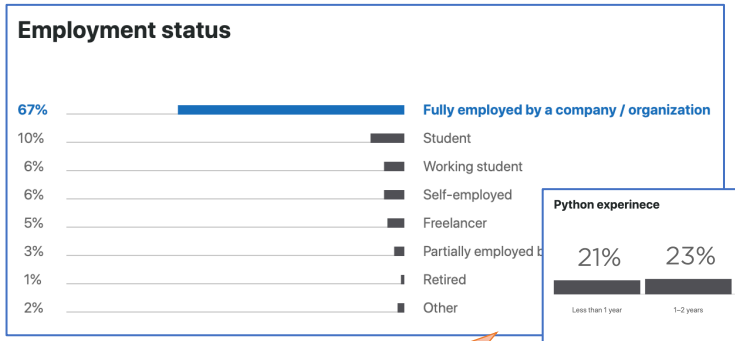
グイド・ヴァン・ロッサム
(1956, オランダ)

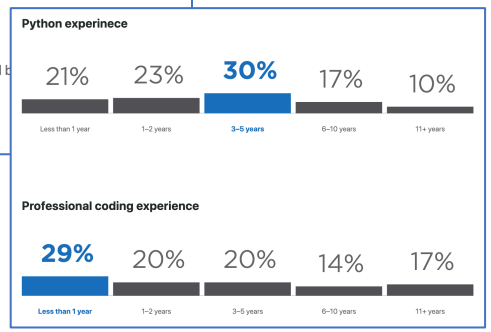
人気ランキングと利用状況

TIOBE Index (https://www.tiobe.com/tiobe-index/)

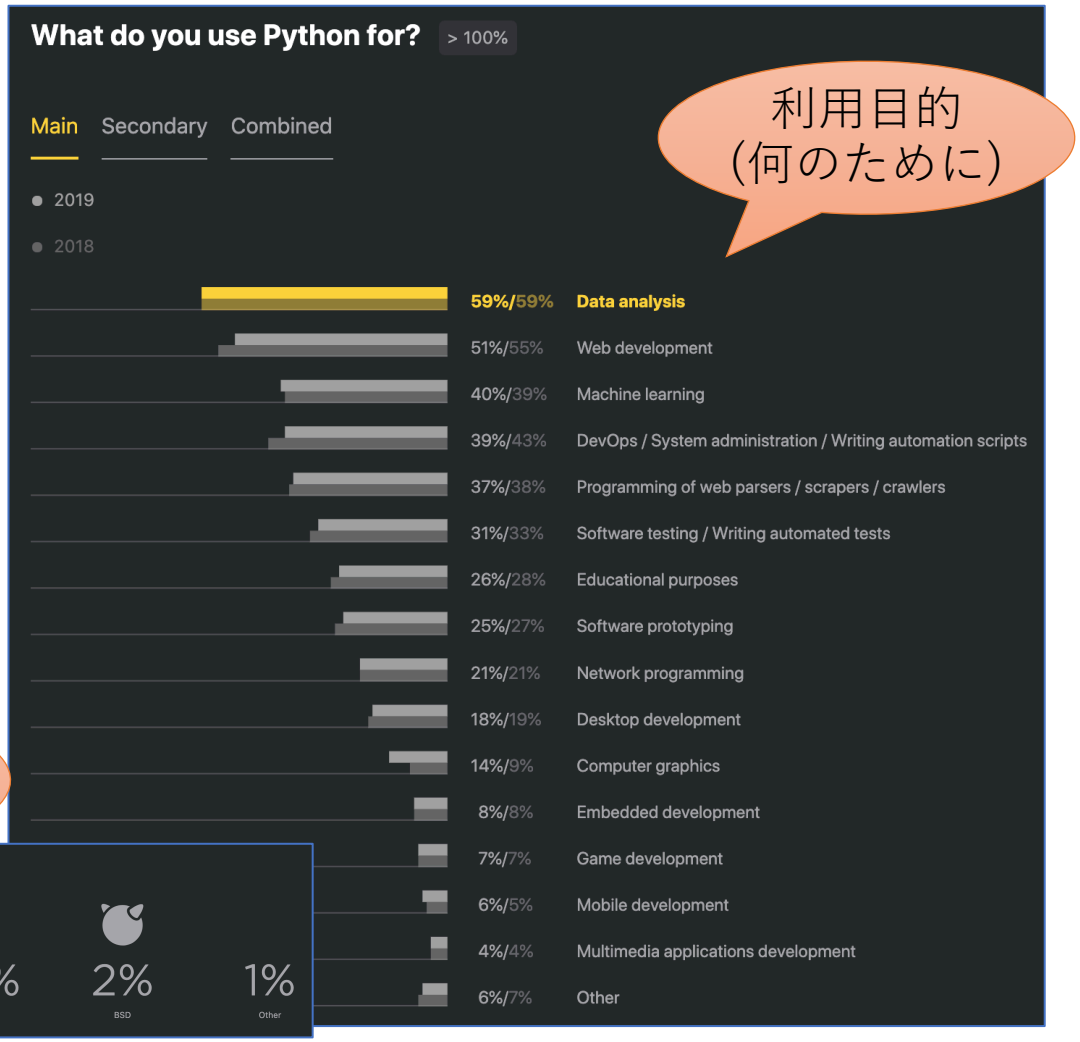


ランキング
(人気度)



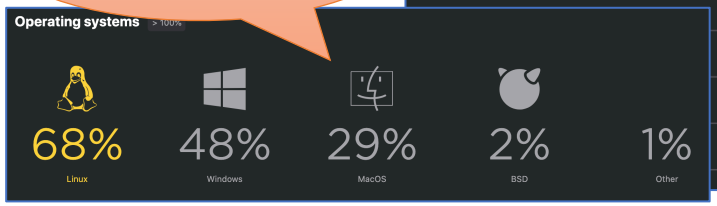


利用者
(誰が)



利用目的
(何のために)

利用方法
(どうやって)



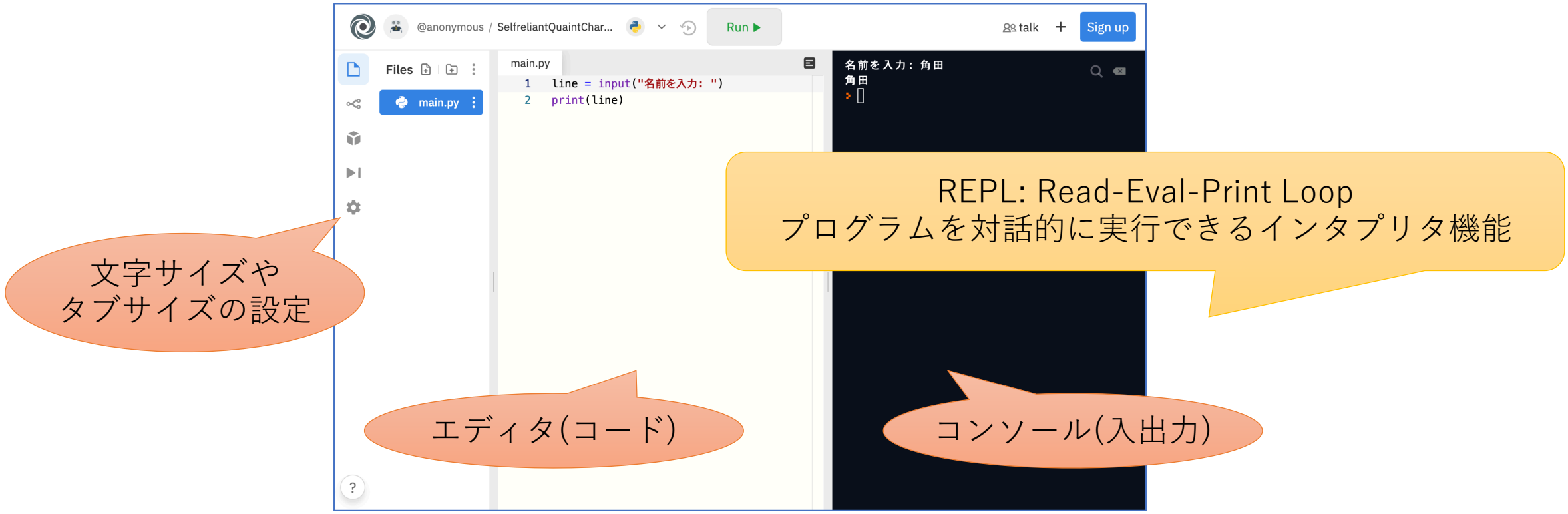
Python Developers Survey (https://www.jetbrains.com/lp/python-developers-survey-2019/)

Pythonの開発環境

- オンライン(学習用、ユーザ登録不要)
  - CodeChef https://www.codechef.com/ide
  - Coding Ground https://www.tutorialspoint.com/codingground.htm
  - codepad http://codepad.org/
  - Geeks for Geeks IDE https://ide.geeksforgeeks.org/
  - ideone https://ideone.com/
  - JDoodle https://www.jdoodle.com/
  - paiza.io https://paiza.io/
  - repl.it https://repl.it/
  - Solo Learn https://code.sololearn.com/
- オンライン(開発用)
  - Colabolatory(※) https://colab.research.google.com/
  - AWS Cloud9 https://aws.amazon.com/jp/cloud9/
  - Codeanywhere https://codeanywhere.com/
- オフライン
  - IDLE(Pythonパッケージ付属の統合開発環境) https://www.python.org/
  - Anaconda(データサイエンス向け開発プラットフォーム) https://www.anaconda.com/

研修会で使う開発環境

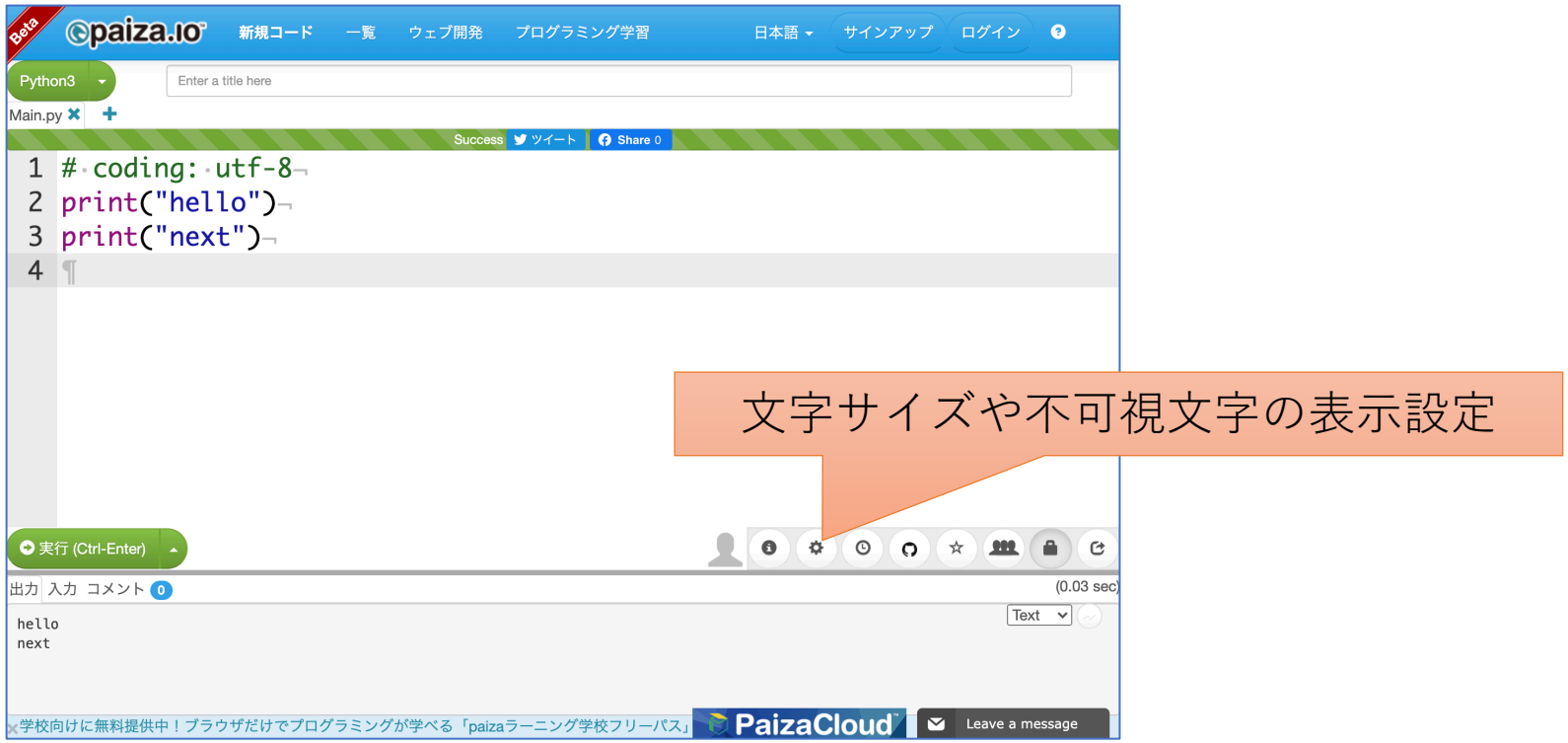
- Repl.it(学習専用のオンラインプログラミング環境)



https://repl.it/

研修会で使う開発環境(予備)

- Paiza.io(学習専用のオンラインプログラミング環境)



https://paiza.io/

Pythonの基礎

入出力

変数と演算

コレクション

入出力

- 画面出力(標準出力)
  - print(値/文字列)
  - print(値/文字列, sep = "区切り文字")
  - print(値/文字列, end = "行末文字")
  - print(値/文字列, file="ファイル名")

- キーボード入力(標準入力)
  - input()
  - input(プロンプト文字列)
  - ※入力された値を文字列として戻す

ファイル出力例

test.py

```
1  myfile = open("output.txt", "w")
2  print("Hello", file=myfile)
3  myfile.close()
```

プロンプトの利用

test.py

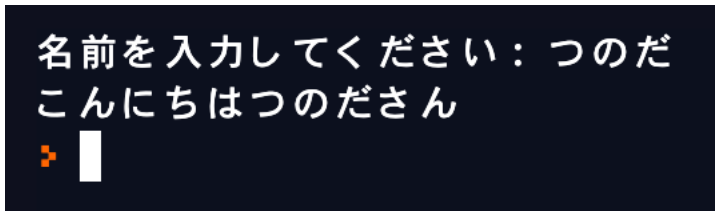
```
1  name = input("名前を入力してください: ")
2  print(name)
```

(解説) Pythonコードの書き方

- コードの記述
  - すべて半角の英数字と記号で記述、大小文字は区別
  - 日本語文字は文字列かコメント内のみ
  - 文字列のクォートはシングル/ダブルどちらでもよい
  - インデント(字下げ、空白2つ/4つ)には意味があるので注意
- コメント(動作に影響しないメモ、機能のon/offテスト)
  - 単一行: #から行末まで
  - 複数行: ''' (3連続クォート)から'''までの範囲
- プログラムファイルの拡張子は".py"

(問題演習) 入出力

- 名前を入力させて、「こんにちは？？さん」と出力するプログラム



変数と演算

- Pythonの変数
  - 基本型は整数(int)、小数(float)、文字列(str)、真偽値(bool)
  - 変数名は英数字とアンダースコア(数字で始めない、大小文字は区別)
    ※実はUnicode文字(日本語)も使えるが、おすすめしない。
- 演算
  - 代数演算(+, -, *, /, **, %, //)、交換/結合/分配法則、優先順位
  - 代入演算(=, +=, -=, *=, /=, **=, %=, //=)
  - 文字列演算(+, *, [])
- 型変換
  - int()、float()、str()、bool()

(問題演習) 型変換

- 以下のプログラムの挙動を修正せよ

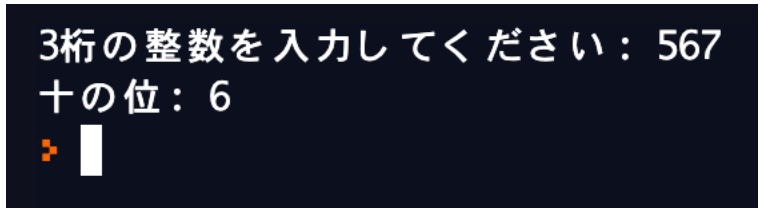
main.py

```
1   v = input("数値を入力: ")
2   print(v + "の2倍は" + v*2 + "です。")
3
```

```
数値を入力: 4
4の2倍は44です。
```

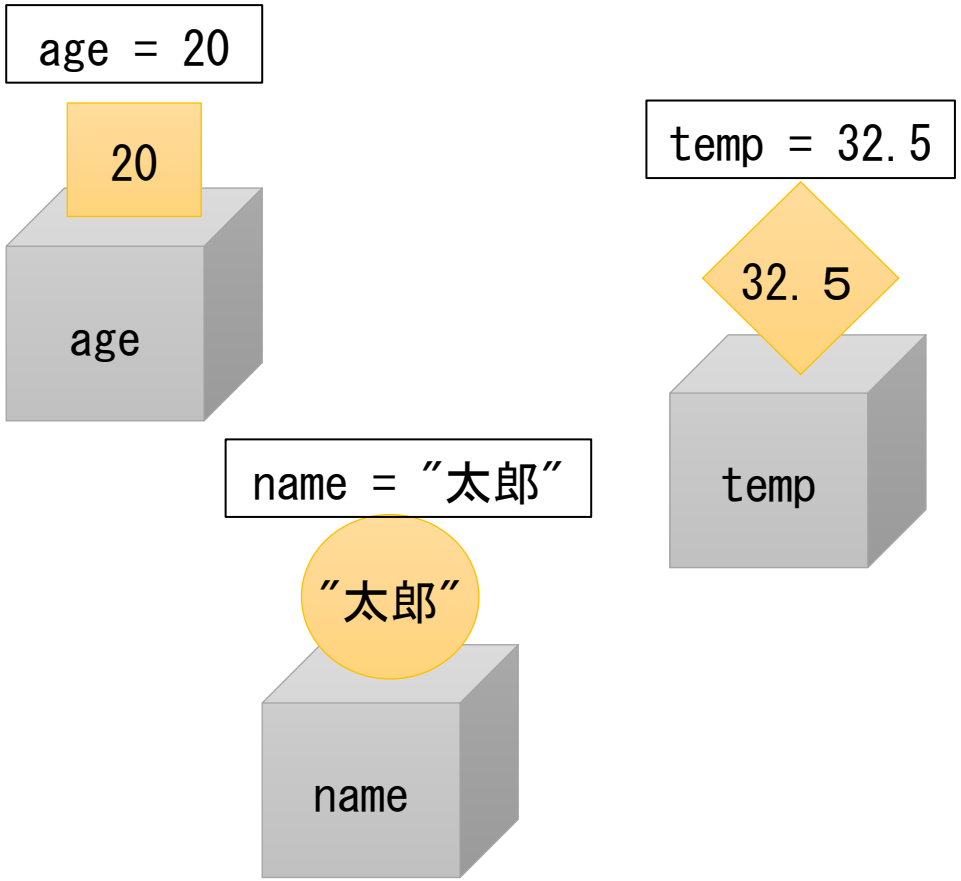
(問題演習) 演算

- 入力した3桁の数値の十の位の値を表示せよ

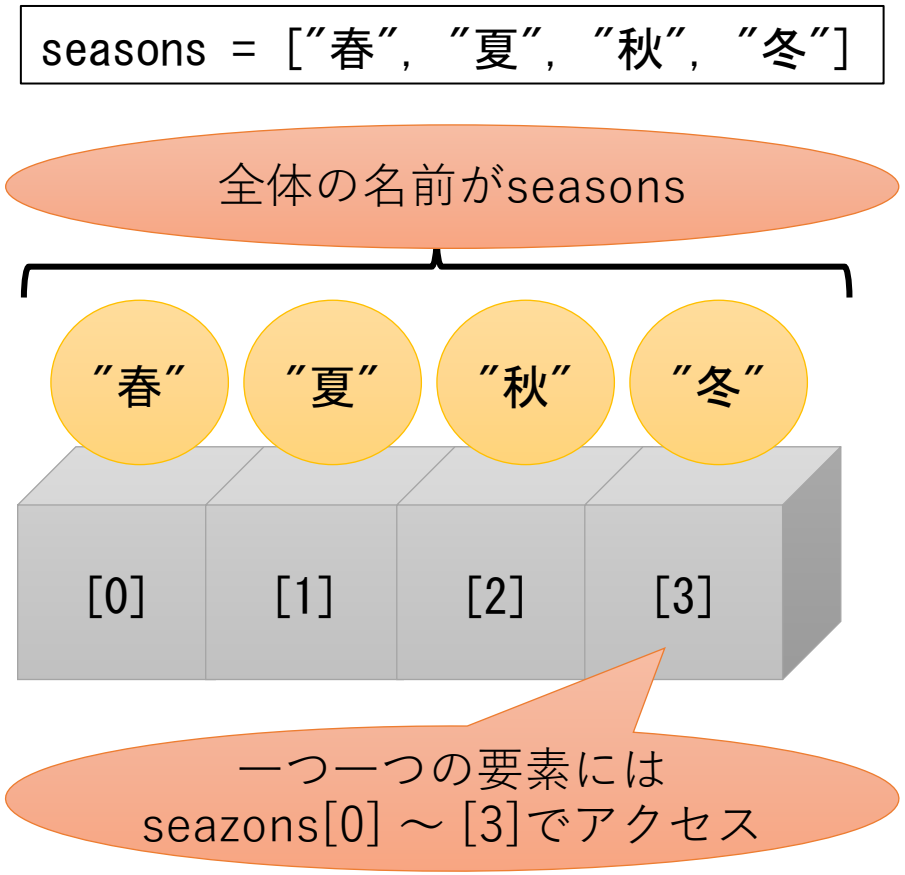


(解説) 変数とコレクション

- 変数のイメージ



- コレクションのイメージ



コレクション(1) リスト

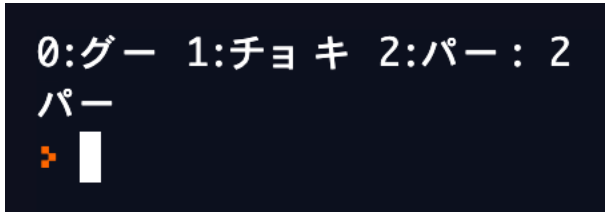
- リスト(list)
  - 作成　　リスト名 = [要素1, 要素2, 要素3, …]
  - 参照　　リスト名[インデックス]
  - 追加　　append(要素)、insert(インデックス, 要素)
  - 削除　　remove(要素)、pop(インデックス)
  - 操作　　len()、sorted()
  - 生成　　split()、range()、[…] * 繰り返し回数
- 注意
  - インデックスは0から開始、要素数をオーバーしないこと
  - 要素の型は一致しなくて良い
  - リスト要素にコレクションを含むことも可能(多重リスト)

コレクション(2) タプル、セット、辞書

- タプル(tuple)
  - 変更不可のリスト(列挙型として使う)
  - 作成　　**タプル名 = (要素1, 要素2, 要素3, …)**
- セット(set)
  - 重複不可のリスト(重複排除、集合演算などで利用)
  - 作成　　**セット名 = {要素1, 要素2, 要素3, … }**
- 辞書(dictionary)
  - キーをインデックスとするリスト(ハッシュ、連想配列)
  - 作成　　**辞書名 = {キー1: 値1, キー2: 値2, キー3: 値3, … }**

(問題演習) リスト

- ジャンケンのプログラム(入力した数に対応した手を表示)



制御構造

逐次実行

繰り返し

条件分岐

(解説) コンピュータとプログラム

- ノイマン型コンピュータ
  - 命令とデータ（＝プログラム）をメモリに記憶させる
  - メモリにはアドレス（番地）がある
  - メモリ上の処理を順番に取り出し、CPUで実行する

メモリ1番地の命令
メモリ2番地の命令
メモリ3番地の命令
メモリ4番地の命令

```
1 print("おはよう")
2 print("こんにちは")
3 print("こんばんは")
4 print("おやすみ")
```

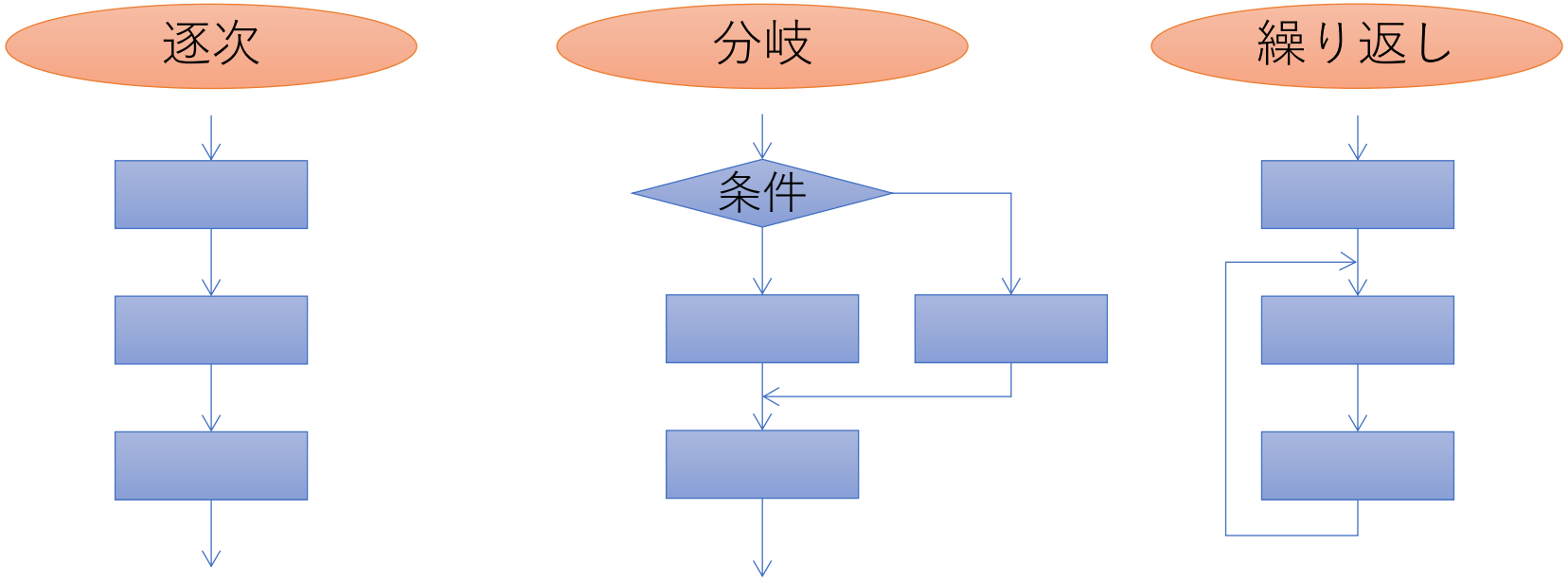
おはよう
こんにちは
こんばんは
おやすみ

上から順に、一つづつ実行される

ジョン・フォン・ノイマン
(1903-1957)

(解説) プログラムの流れを制御する

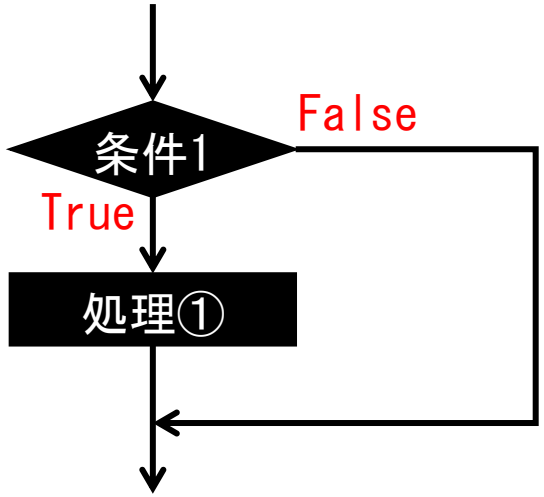
- 構造化プログラミング



どんなプログラムでも３つの流れの組み合わせで表現できる

エドガー・ダイクストラ
(1930-2002)

条件分岐(1)



- if文の書き方

  if 条件:

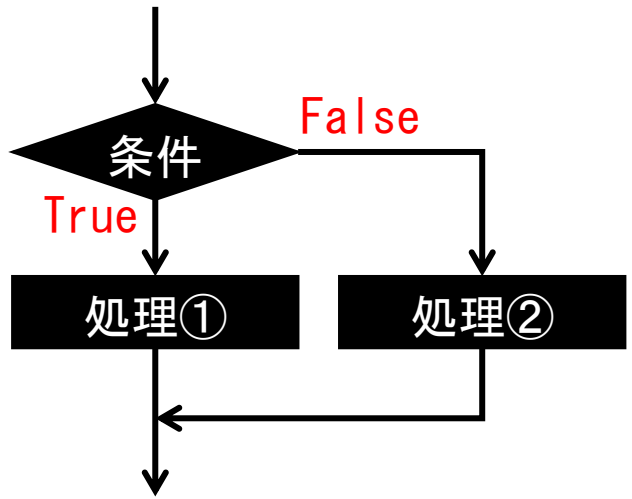
  　　条件が成立する場合(True)の処理①

  ※コロンを忘れずに

  ※必ず先頭にインデントをいれる

- 条件の記述方法
  - 比較演算　　==, !=, >, >=, <, <=, is, is not, in, not in
  - 論理演算　　and, or, not
  - 戻り値(bool型)True/False

条件分岐(2)

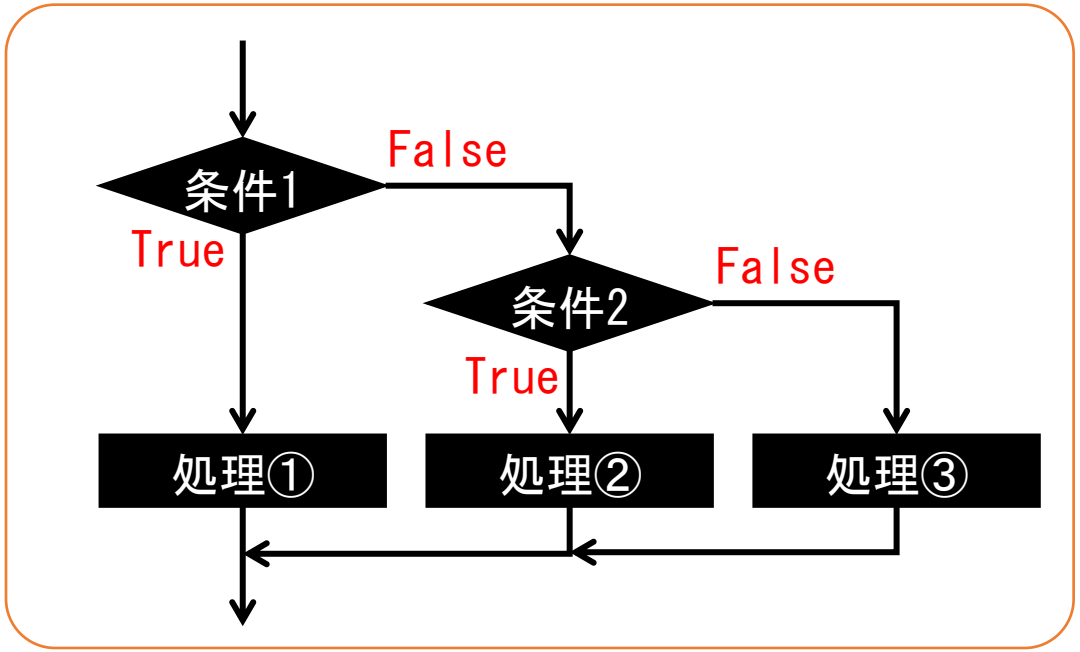


- if～else文の書き方

```
if 条件:
    条件が成立する場合(True)の処理①
else:
    条件が成立しない場合(False)の処理②
```

※コロンを忘れずに
※必ず先頭にインデントをいれる

条件分岐(3)



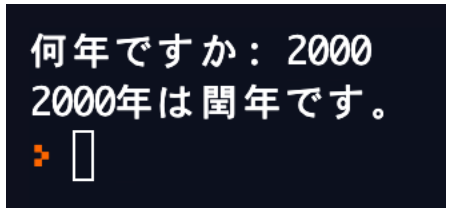
- if～elif～else文の書き方

```
if 条件1:
    条件1が成立する場合の処理①
elif 条件2:
    条件1が成立せず、条件2が成立する場合の処理②
else:
    条件1も条件2も成立しない場合の処理③
```

※コロンを忘れずに
※必ず先頭にインデントをいれる

(問題演習) 条件分岐

- 入力した西暦年が閏年か平年かを答える



※閏年の定義(グレゴリオ暦: 1582年10月15日以降)

① 400で割り切れる年は閏年
② ①以外で、100で割り切れる年は平年
③ ①②以外で、4で割り切れる年は閏年
④ ①②③以外は、平年

繰り返し(1)

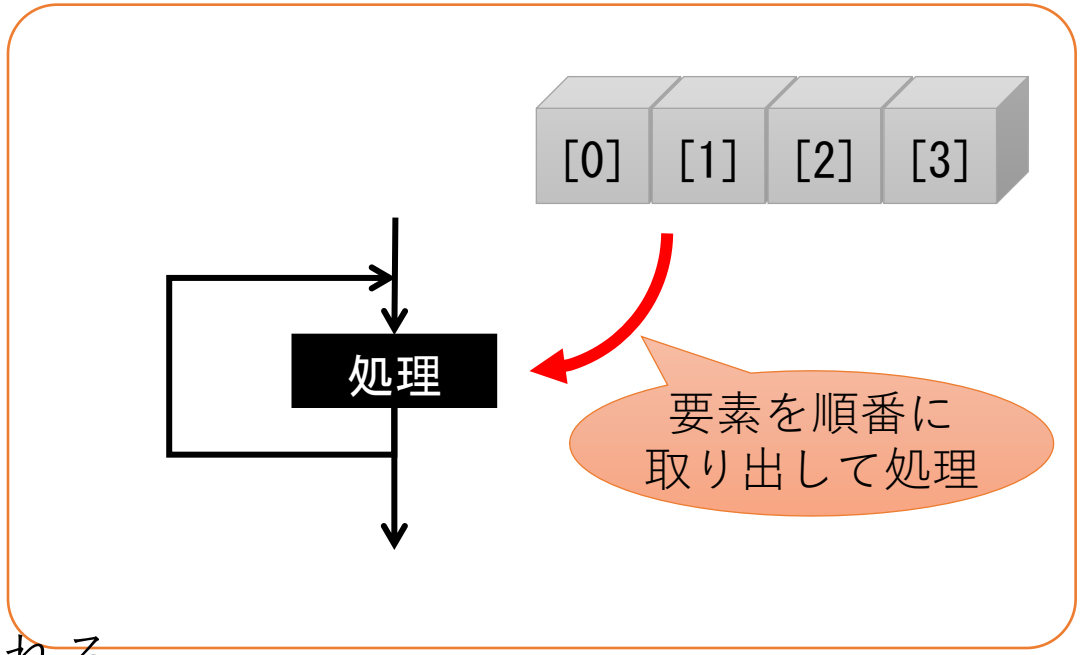
- for文(コレクションに対する繰り返し)

```
for 変数 in コレクション:
    繰り返す処理
```

※コレクションの要素が順番に**変数**に代入される
※コレクションの要素数だけ繰り返す

```
for インデックス変数, 変数 in enumerate(コレクション):
    繰り返す処理
```

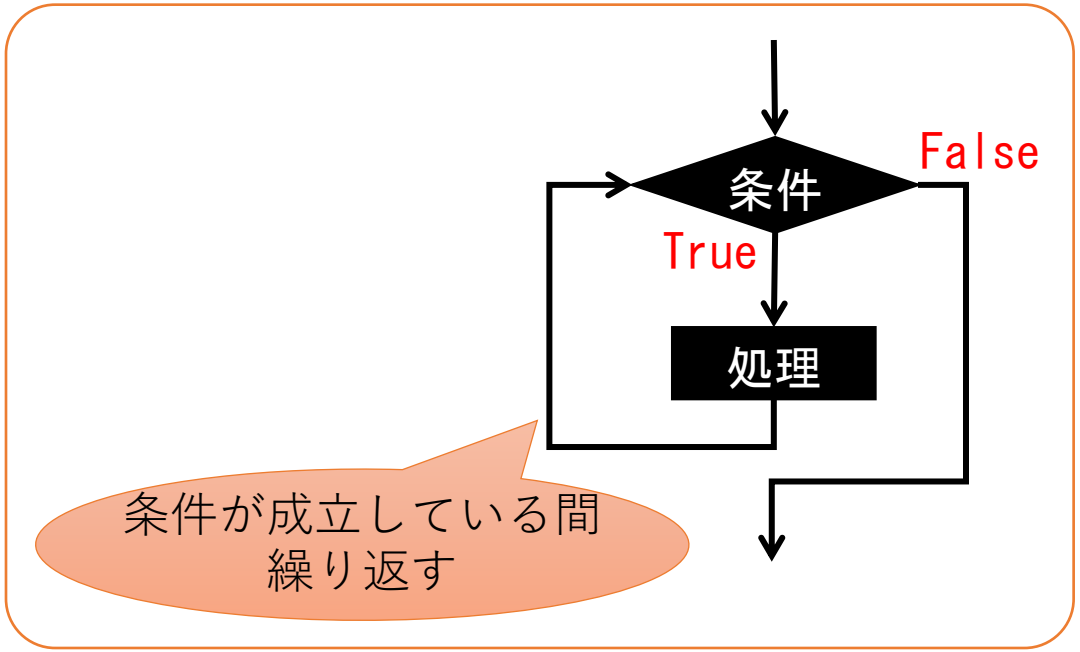
※インデックスが必要な場合



繰り返し(2)

- while文(条件による繰り返し)

```
while 条件:
    繰り返す処理
```



※通常、初期化と条件要素の更新が必要(ないと無限ループになる)

(問題演習) 繰り返し

- 九九の表を出力する

main.py

```
1   for y in range(1, 10):
2       for x in range(1, 10):
3           print(x * y)
4
```

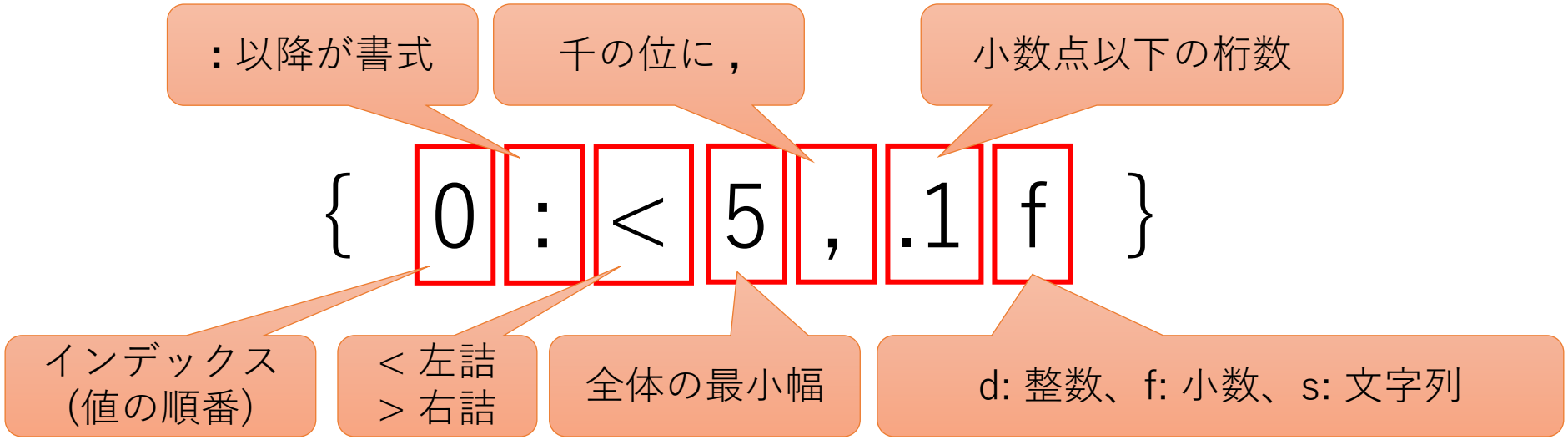
修正

```
 1  2  3  4  5  6  7  8  9
 2  4  6  8 10 12 14 16 18
 3  6  9 12 15 18 21 24 27
 4  8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
 7 14 21 28 35 42 49 56 63
 8 16 24 32 40 48 56 64 72
 9 18 27 36 45 54 63 72 81
```

(解説) 画面表示の書式指定

- formatメソッド

```
print("書式指定文字列".format(値1, 値2, … ))
```



(問題演習) 条件による繰り返し

- 数当て(当たるまで繰り返す)

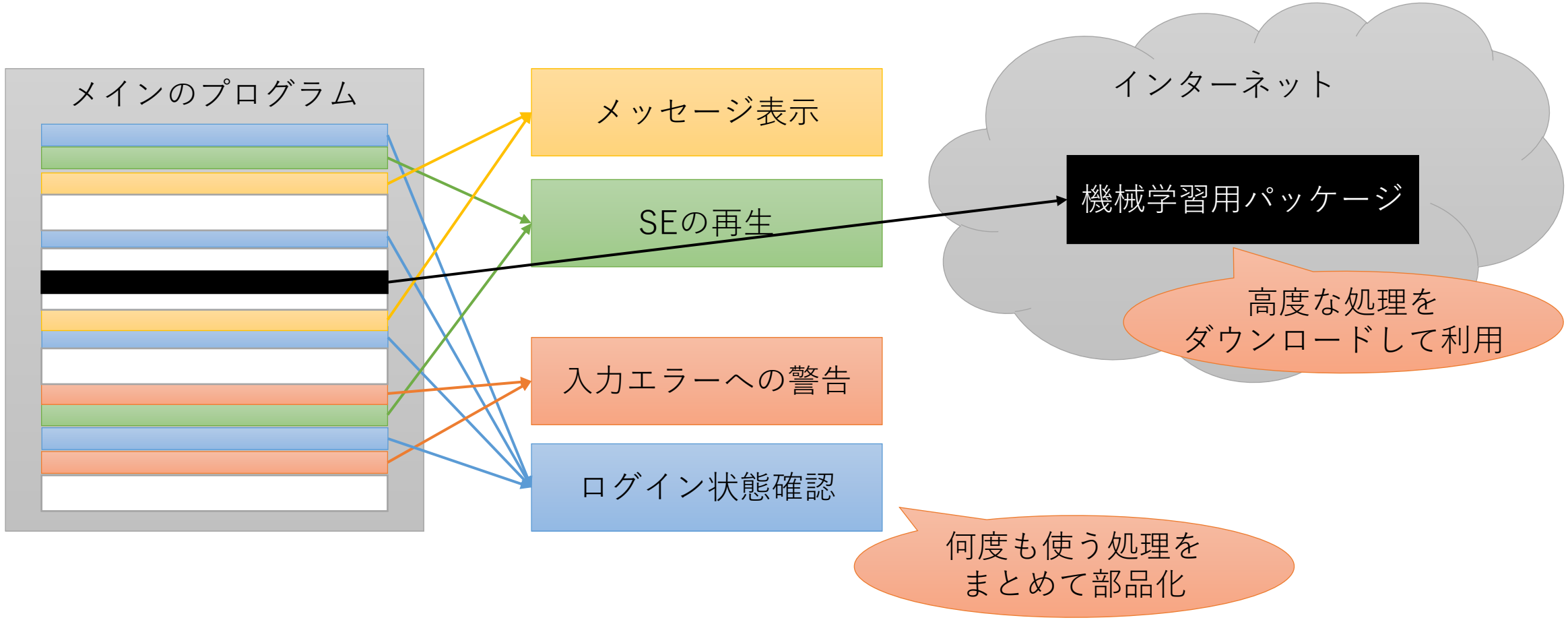
関数

関数の定義と呼び出し

変数のスコープ

(解説) プログラムの部品化

- モジュール(関数、オブジェクト、パッケージ)の利用



関数定義と呼び出し

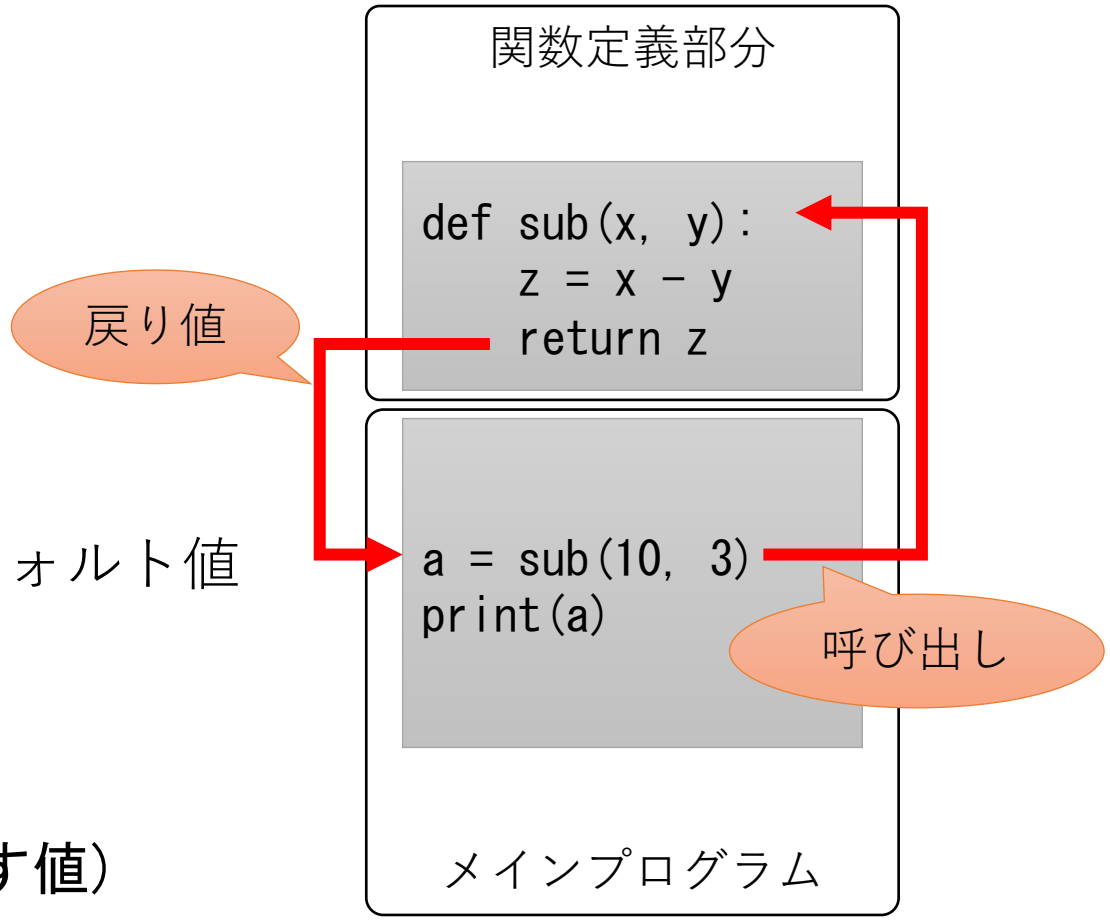
- 関数の定義

```
def 関数名(引数):
    関数内の処理
    return 戻り値
```

※呼び出しよりも前に定義する
※引数は ***引数** でリスト、 **引数=値** でデフォルト値

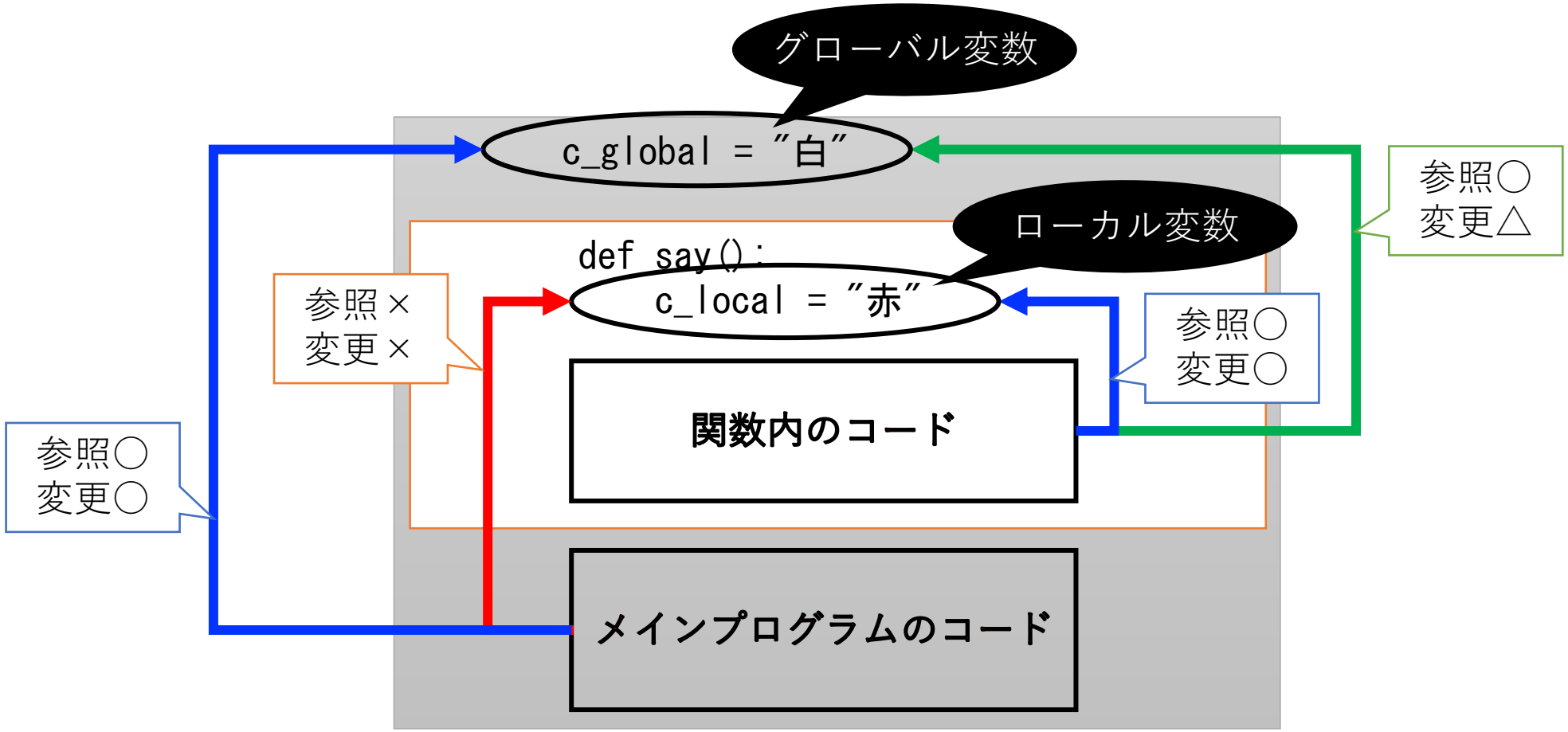
- 関数の呼び出し

戻り値を受け取る変数 = 関数名(引数に渡す値)



変数のスコープ

- グローバル変数とローカル変数



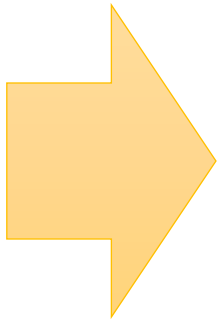
(問題演習) 関数

- 関数定義の部分を記述せよ

main.py

```
1
2
3
4
5   team = ["勇者", "戦士", "魔法使い"]
6
7   enemy = "スライム"
8
9   for p in team:
10      battle(p, enemy)
11
```

？



```
勇者はスライムを攻撃した
スライムに10のダメージを与えた
戦士はスライムを攻撃した
スライムに10のダメージを与えた
魔法使いはスライムを攻撃した
スライムに10のダメージを与えた
>
```

オブジェクト

クラスとインスタンス

メソッドとプロパティ

継承

クラスとインスタンス

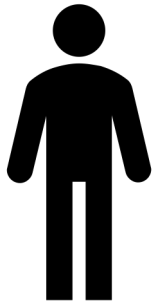
- 設計図と実体

インスタンス

性質や機能を
具体的に決めた実体

Playerクラス

名前:
職業:
歩く()
闘う()

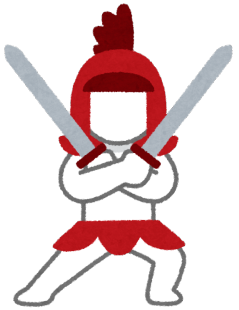


生成

生成

生成

名前: 太郎
職業: 勇者
歩く(足が早い)
闘う(勇者の剣で)



名前: 次郎
職業: 戦士
歩く(足が遅い)
闘う(二刀流で)

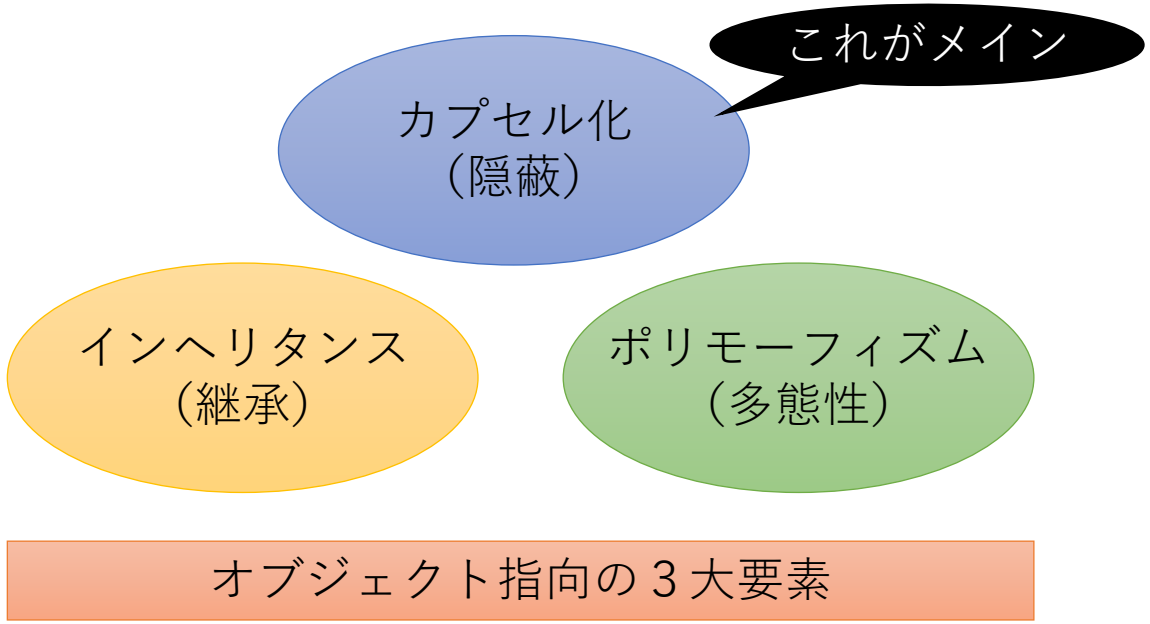
名前: 花子
職業: 魔法使い
歩く(空を飛ぶ)
闘う(全体攻撃)

クラス

Playerの持つ性質や
機能の設計図

(解説) オブジェクト指向

- ダイクストラの「構造化プログラミング」の進化形
- プログラムの機能や状態を部品（オブジェクト）の中に閉じ込め、通知（メッセージ）を送ることでオブジェクトを操作する。



メソッドとプロパティ

- オブジェクトの考え方

プログラムで扱う「もの」

オブジェクトの性質・状態

オブジェクト ＝ 関数(メソッド) ＋ 変数(プロパティ)

オブジェクトの持つ機能・処理

- プログラムの実装

①クラス定義

```
class クラス名:
    def __init__(self, 引数):
        初期化処理

    def メソッド名(self, 引数):
        機能の処理
```

②インスタンス生成

```
インスタンス変数名 = クラス名()
```

③メソッド呼び出し、変数操作

```
インスタンス変数名.メソッド名(引数)
インスタンス変数名.変数名
```

(補足説明) 用語など

- コンストラクタ(インスタンス生成時に一度だけ実行、初期化)
  __init(self, **引数**)__
- インスタンス変数(インスタンスに固有)
  self.**変数名**
- クラス変数(クラスに共通)
  クラス名.変数名
- クラスメソッド
  @classmethod で修飾

オブジェクトの保護

- 外部からのアクセス制限
  - 変数名やメソッド名の前に"__"(アンダースコア2つ)をつける

- プロパティへの「アクセサ」
  - 値の参照や変更を専用のメソッド経由で行う

値の参照	@property
値の変更	@プロパティ名.setter
プロパティの削除	@プロパティ名.deleter

※オブジェクトへの不正な操作(バグ、チート)を制限する

継承

- 親クラス(スーパークラス)の性質を受け継ぐ子クラスを定義
  - クラス定義

    class 子クラス名(親クラス名):

        処理内容
  - 親クラスのメソッドやプロパティを利用可能
  - 子クラス独自のメソッドやプロパティを追加可能
- オーバーライド
  - 親クラスのメソッドを子クラスのメソッドで上書き(オーバーライド)
  - 子クラスのメソッドが有効になる
  - 親クラスのメソッド呼び出しも可能

    super().メソッド名()

(問題演習) オブジェクト

- Enemyクラスのhit()メソッドを定義せよ

main.py

```
1  import random
2
3  class Enemy:
4      def __init__(self, n):
5          self.name = n
6          self.hp = 10
7
8      def hit(self):
9          
10 
11 
12 
13 
14
15 e1 = Enemy("スライム")
16 for i in range(0, 5):
17     if e1.hp > 0:
18         e1.hit()
```

?

```
スライムに4のダメージ
スライムに0のダメージ
スライムに3のダメージ
スライムに5のダメージ
スライムを倒した
```

ライブラリの利用

標準ライブラリ
matplotlib + NumPy
Beautiful Soup
Tkinter

モジュールの利用方法

- モジュールの組み込み
  - モジュール全体　　import モジュール名 as 別名
    (モジュール名.属性名で呼び出し)
  - 特定属性を指定　　from モジュール名 import 属性名 as 別名
    (属性名/別名で呼び出し)

※「モジュール」… 外部ファイル(*.py)に定義された機能
※「パッケージ」… 複数のモジュールで構成される機能群
※「ライブラリ」… プログラムの外部で定義された機能(曖昧)
※「標準ライブラリ」 … Python本体と共にインストールされる機能
(注) 標準ライブラリ以外は環境へのインストールが必要

モジュールの利用例(標準ライブラリ)

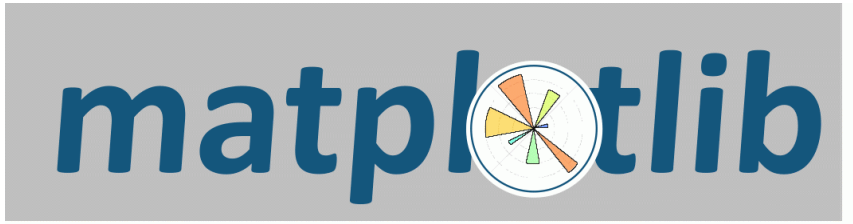
- 乱数モジュールの利用(import random)
  - 0から1未満の乱数(小数)　　random.random()
  - 指定した範囲の乱数(整数)　random.randint(**最小値**, **最大値**)
- 時間モジュールの利用(import time)
  - エポック秒の取得　　time.time()

    ※エポック秒(UNIX秒) … 1970年1月1日午前0時0分0秒からの経過秒数
  - 処理の一時停止　　time.sleep(**秒数**)
- 数学関数モジュールの利用(import math)
  - 円周率　　math.pi
  - 三角関数　　math.sin(**角度**) …

グラフ描画ライブラリ matplotlib

- 概要／特徴
  - NumPyのためのグラフ描画
  - 様々な種類のプロット(折れ線/棒グラフ/散布図/等高線 …)
  - 複数グラフの作成、画像保存、3次元プロット、アニメーション
- 使い方
  - 読み込み

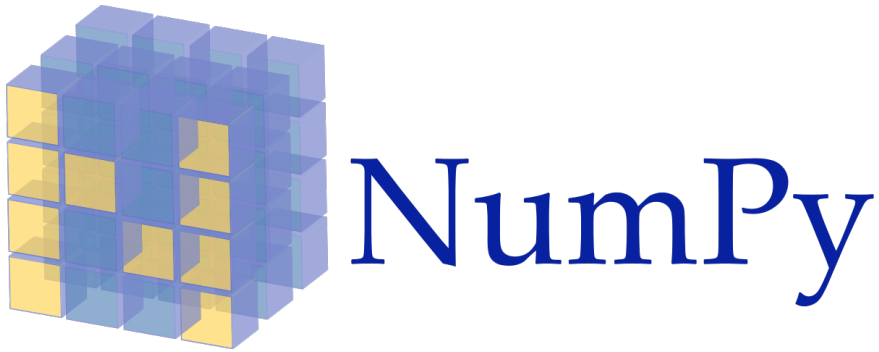
```
import matplotlib.pyplot as plt
```



科学計算ライブラリ NumPy

- 概要／特徴
  - 効率的な数値計算、行列・ベクトル計算
  - Pythonの弱点(計算処理が低速)を補う
  - 機械学習・深層学習に必要な大規模なデータ処理
- 使い方
  - 読み込み

```
import numpy as np
```



HTMLライブラリ BeautifulSoup

- 概要／特徴
  - HTML/XMLの処理
  - Webからのデータスクレイピング(データ抽出)
  - 機械学習のための学習用データ作成
  - 意思決定のための業務データ抽出・分析
- 使い方
  - 読み込み

```
from bs4 import BeautifulSoup
import requests
```

GUIライブラリ Tkinter

- 概要／特徴
  - tk(GUIアプリ用ツールキット)のPython用ライブラリ
  - 各種ウィジェットの生成／処理
    - フレーム、ラベル、ボタン、チェックボックス、フォーム …
  - ループとイベント処理
- 使い方
  - 読み込み

```
from tkinter import *
from tkinter import ttk
```

最後に

Pythonで仕事するには

もっと勉強するには

落穂

- 取り上げなかった重要な項目
  - 例外処理
    - エラー制御、リカバリ　　→ Webスクレイピングに必須
  - 並列処理
    - スレッド、スケジューラ　　→ ネットワーク通信
  - 文字列処理
    - 大文字、小文字の変換、正規表現　　→ 自然言語処理

学習素材

- オンラインの学習サイト
  - Paiza Learning (https://paiza.jp/works/)
    - 基礎編は無料、開発環境あり
  - ドットインストール (https://dotinstall.com/)
    - 一部無料、開発環境なし
  - progate (https://prog-8.com/)
    - 一部無料、開発環境あり
  - Python-izm (https://www.python-izm.com/)
    - Python専用教材、無料、開発環境なし

※高額な有料サイトに注意！！