

# Pythonプログラミング 勉強会用

藤堂 健世

2020/05/19（作成）

# Pythonの実行環境について

■Python事前講義で行ったようにGoogle Colaboratoryを利用します

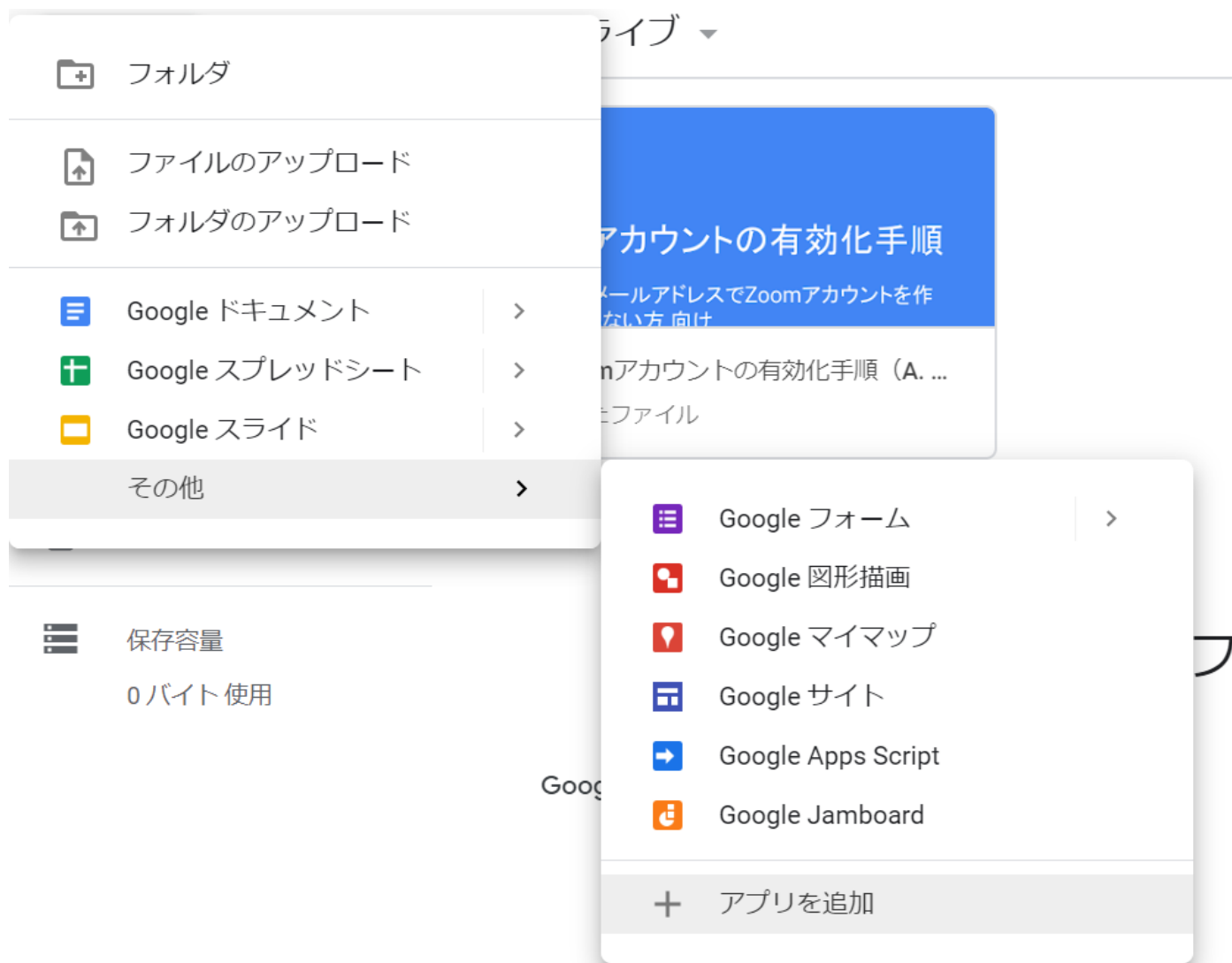
（勿論自前の環境を利用しても構いません。ただし後述の通りVersionはGoogle Colaboratoryに準拠します）

Googleアカウントがあれば（Google Colaboratory）を起動する事ができます。

■Google Colaboratoryのインストール方法

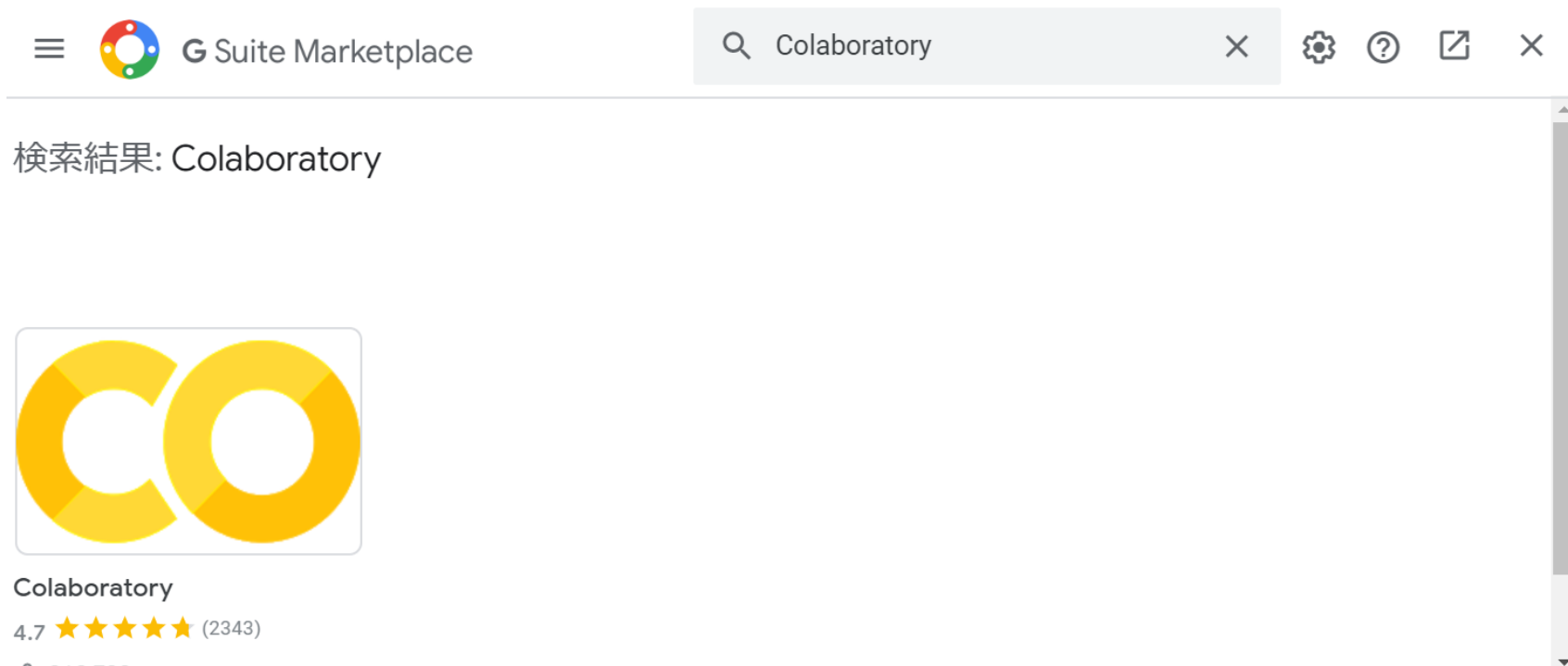
「新規」→「その他」→「アプリを追加」を選択

# Pythonの実行環境について



# Pythonの実行環境について

検索部分に「Colaboratory」と入力してアプリをインストールすると利用可能になる。「新規」→「その他」の中に「Colaboratory」が利用可能になる



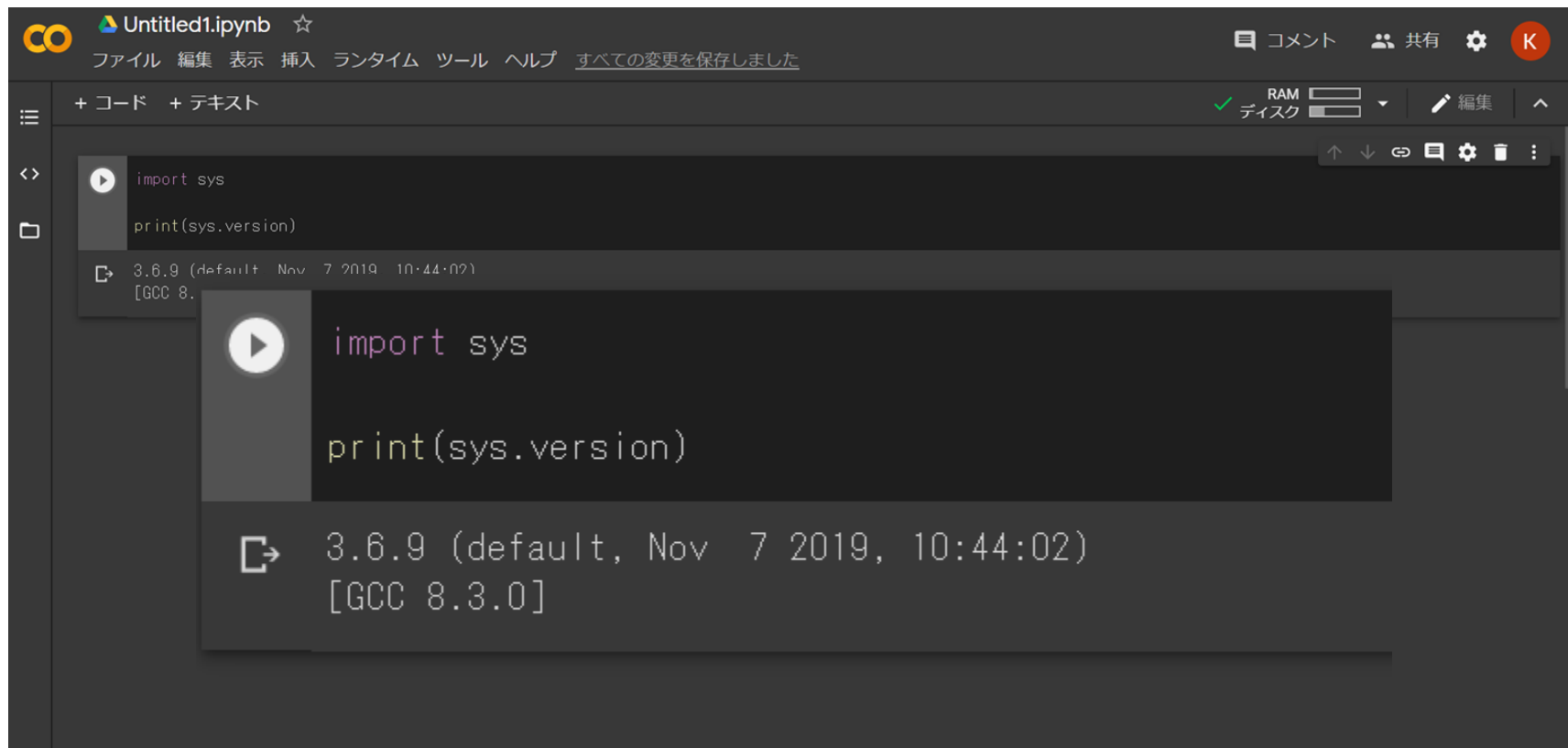
# Pythonの実行環境について

## ■Colaboratoryの起動を確認する

```
import sys
```

```
print(sys.version)
```

で現在使われているPythonのバージョンが確認できる





# Pythonについて

- 1991年に登場したプログラミング言語
- フリーソフトかつオープンソース
- クロスプラットフォーム
- 字下げ（インデント）を強制する

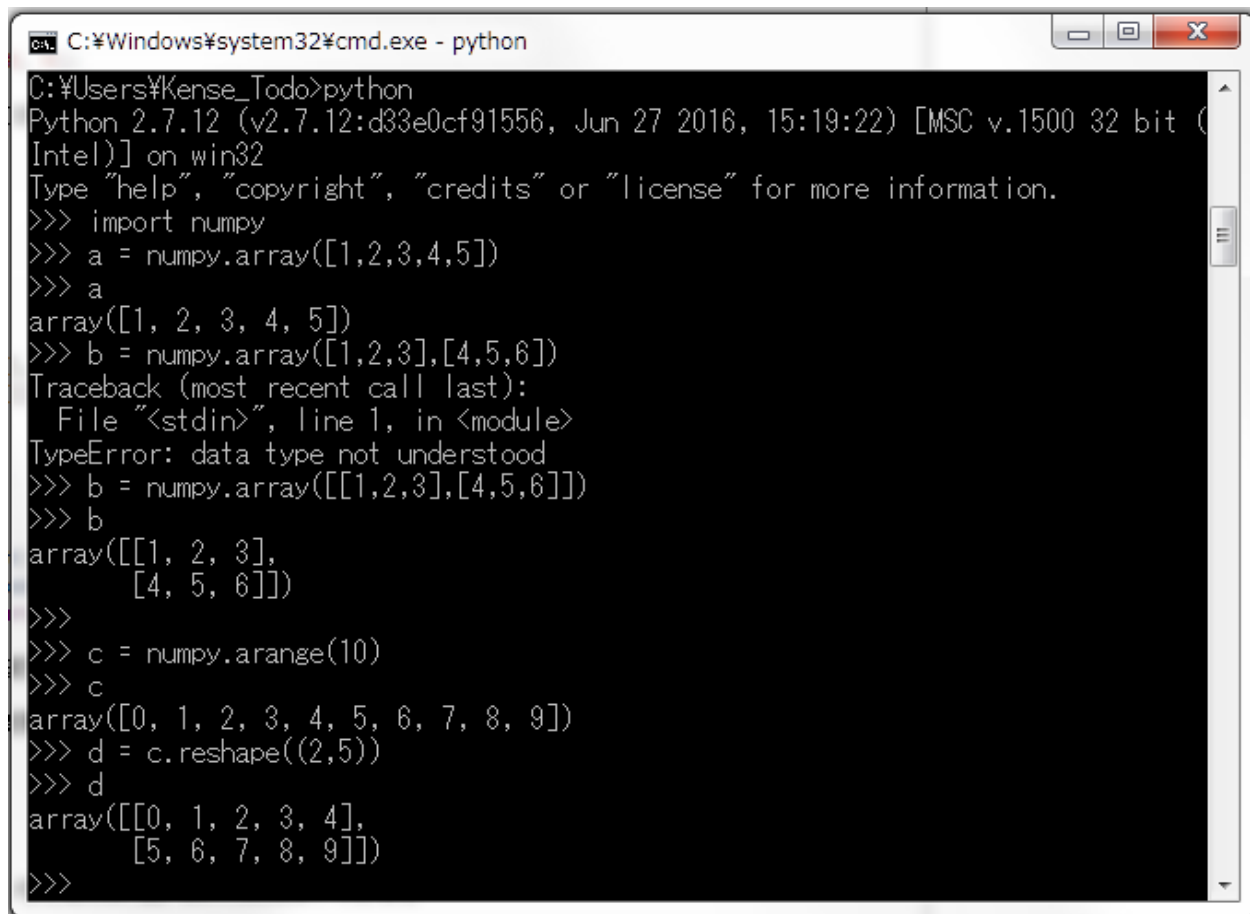
```
def factorial(x):  
    if x == 0:  
        return 1  
    else:  
        return x * factorial(x - 1)
```

```
def factorial(x):  
    if x == 0: return 1 else:  
        return x * factorial(x - 1)
```

C言語のようには  
かけません。

- ちなみに、YoutubeやDropboxもPythonで書かれている

# Pythonはスクリプト言語



```
C:\Windows\system32\cmd.exe - python
C:\Users\Kense_Todo>python
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> a = numpy.array([1,2,3,4,5])
>>> a
array([1, 2, 3, 4, 5])
>>> b = numpy.array([1,2,3],[4,5,6])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: data type not understood
>>> b = numpy.array([[1,2,3],[4,5,6]])
>>> b
array([[1, 2, 3],
       [4, 5, 6]])
>>>
>>> c = numpy.arange(10)
>>> c
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> d = c.reshape((2,5))
>>> d
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>>
```

## スクリプト言語の特徴

対話するかのよう  
に、プログラミング言語  
を入力していく

予めプログラミングを  
作っておいて実行する  
ことももちろん出来る

が、こんな感じで  
デバックや動作確認  
を行ったり出来る。

# JAVA（コンパイル言語）との違い

	シェル・スクリプト	スクリプト言語	コンパイル言語
利便性	△ （計算は得意じゃ無い）	◎ （言語の書き換えが容易）	○ （コンパイル後は書き換え面倒）
計算速度	○	×（ものによる）	◎
使い勝手	？ （プログラムを何回も実行したりとか何かの処理）	◎ （OS・マシンの依存性なし）	△ （コンパイル必要） （マシンとの依存性あり）
用途	プログラムの処理	数値計算	本実験
種類	Bshell Cshell Zshell Bash	Python Ruby Perl JavaScript	JAVA C C++



# Python2系と3系について

Pythonは2系と3系でだいぶ変化がありました

- ・ Print文の関数化

```
def print(*args, sep=' ', end='\n', file=None)
```

↑どうやらこんな関数らしいです

- ・ 比較演算子(<, >, <=, >=)は、そのオペランドが自然な順序付けを持たない場合  
TypeErrorを送出する

2系だと仕様上Trueをかえすことがあった

- ・ 1/2のような式はfloatを型を返す。

少数点以下切捨て場合は1//2のような式にすること

- ・ ユニコードへ

教科書によっては、2系を利用してる場合があります。


(最近はほぼ3系。3年前とか4年前は2系と共存している)

# Print文

- それでは書いてみましょう。→print (“文字列”)

2系では（）がなくとも表示させることが可能でした。今は昔の物語です。

```
C:\Users\Kense_Todo>python
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit
Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print "Hello, World!"
Hello, World!
```



- Print文に代入したモノを入れ込みたい場合は・・・

```
>>> a = 10
>>> b = "pens"
>>> print ("Those are {0} {1}.format(a,b))
File "<stdin>", line 1
    print ("Those are {0} {1}.format(a,b))
                                ^
SyntaxError: EOL while scanning string literal
>>> print ("Those are {0} {1}".format(a,b))
Those are 10 pens
```

- .format() を使いましょう

○ “”を忘れないように

# Print文

- C言語っぽく書くことも出来ます。

```
>>> print ("Epoch %i/%s" % (a,b))  
Epoch 10/pens
```

- JAVAっぽく書くことも出来るわけだ . . .

```
>>> print ("Epoch" + a + "aiueo" + b)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: cannot concatenate 'str' and 'int' objects  
>>> print ("Epoch" +str(a) +"aiueo"+b)  
Epoch10aiueopens
```

- +でつなぐときは、print文の中で数値型、文字列型を揃えないと叱られます。
- print(変数)で、変数内のオブジェクトが呼び出されます

# Print文余談

## コメントアウト

- `#This program is hogehoge...`
- ハッシュタグで対応
- `"""hogehogehoge"""`
- 行をまたぐときはダブルクォーテーションを3つ

# 数値型・文字列型

- 「変数名 = 値」 で作成可能
- 他のプログラミング言語と異なり、データの型を宣言しなくて良い！  
(動的型付け)
- 数値型
  - $a = 1$
  - $b = 3.14$
  - $c = 2 + 3j$
  - 四則演算 (+, -, \*, /, %, \*\*) → (和、差、積、除、剰余、乗)
  - 2.x系と.3x系では除算の扱い方が異なるので注意  
(3系で小数点切り捨ては//を行う)

# 数値型・文字列型

- 文字列型・・・“”か’でくくった文字を格納
  - `text = "this is string"`
- 数値型・文字列型を扱う際の注意（それぞれ確認せよ）
  - `10 + 20`
  - `"10" + "20"`
  - `10 + "20"`
- 数値型→文字列型、文字列型→数値型
  - `str(10), int("20")`
  - （floatもあるだよ）
- `len(str)`・・・文字列strの文字列長を返します

# リスト(list) (型の1つ)

- Pythonのリストには1つのリストに数値、文字列、リストなど様々な型のデータを格納することができる
  - `l = ['a', 1, 2, 3, 4, ['a', 3]]`
  - `a_in_list = 'a' in list` でリスト内で要素があるか確認
- 要素の差し替え、追加、削除が可能
  - `l[1:4] = ['taro', 'jiro']`
  - リスト.append(オブジェクト)
  - リスト.remove(オブジェクト)
- 他にも機能があります。
  - 並び替え、指定の要素の削除、最大、最小

# リスト(list)

- リスト内の要素（オブジェクト）へのアクセス方法

- 至って簡単
- `list = [1,2,3,4,5]`
- `list[0]`
- `list[1:4]`

- リスト内の要素の書き換え方

- こいつも簡単
- `list[1] = 3`
- こんな感じで書き換えられます



# タプル(tuple) (型の1つ)

- リストと少し似ている
  - tuple = (1,2,3,4,"a")
  - カッコ()でくるだけ
- 要素へのアクセス方法は同じ
- 要素の差し替え、追加、削除は出来ない
  - 計算速度がリストより若干早い
  - マップのキーにできる

# ディクショナリ（型の1つ）

- 辞書型（ディクショナリ）はキーバリューストアでデータを保存する

- `D = {'key1': 'value1', 'key2': 'value2'}`
- キーバリューストア：郵便番号みたいなもん
- キー：930-0953    バリュー：富山県富山市秋吉

- データの取得

- `ディクショナリ[キー]`
- `ディクショナリ.get(キー)`

- データの追加

- `ディクショナリ[キー] = バリュー`

# ディクショナリ

- データの削除

- `del ディクショナリ[キー]`

- タプルとの組合せ

- `Diary = {}`
  - `key = ('Todo', '08-11')` ←ここがtuple型
  - `diary.key = '70kg'`
  - `Print (diary)`
  - タプル型をkeyとして扱っている

# 道具としての関数

- Pythonにおける関数について
  - 例えば文字列の文字数を測りたいとき
  - `len('python')`としたとき、6が戻り値として帰ってくる
  - よく使う関数
    - `len`
    - `range`
    - `str`
- あとPythonには、データ型に関数(メソッド)を持っていて呼び出すことができる

# 関数の作成

- 次のように書きます

- 引数や戻り値は絶対ではない

def 関数名(引数):  
 命令文  
 return 戻り値



- どこかで作っておいて何度も使う命令は関数を作っておいたほうが良い

- Pythonは関数を別のファイルに作っておいて呼び出すことも可能(モジュールのインポート)

- import プログラムファイル名
- プログラムファイル名.関数名(引数)

# 関数の作成の注意

- 関数の中と外で変数は一致させない
  - というのも、関数内の変数と関数外の変数は別物だから
  - スコープの範囲を気をつけてください

```
i=3
def func():
    print(i)

func()
3
```

```
i=3
def func():
    i=5
    print(i)

func()
5
Print(i)
3
```

**cf:copy/deepcopy**

→この辺の話はややこしいので、適に調べて見てください。今回の問題演習では出てきません。

# If文

- 分岐処理を行うときに利用
- 基本的な形

if 条件:

    処理（インデントを下げること）

- If~else文

if 条件:

    処理A

else:

    処理B

# If文

- 条件には、比較演算子を利用する

- ==

- !=

- >=

- <=

- >

- <

- 型をまたがった比較は出来ないので注意

- and or not

- 複数の条件を組合せて複雑な「条件」を作れる

- if not (a ==1 and 'z' in b) or len(b) !=3:



# for文

- for i in range(数値) :
  - 基本形はこの形です！
  - 数値の回数を繰り返すということになります。（0から数値-1の数）
  - iには、実行中の回数を格納している
- for i in range(数値A,数値B) :
  - 数値Aから数値B-1まで繰り返す
- for i in range(数値A,数値B,-1):
  - 数値Aから数値Bまで逆順で繰り返す
- for i in リスト:
  - リストの中身を繰り返す

# for文

- for index , elem in enumerate(リスト) :

- このタイプはリストの要素を 1つ 1つ取得しながら要素それぞれに処理を行い時に利用する
- この場合index にループカウンタが格納され
- Elemにリストの要素が格納できます

- Break

- 繰り返し処理を終了する処理です
- If文と組み合わせて利用します。

# 基礎編問題

## 問1

- Pythonに`8//3` を計算させると、いくらになるか、確認しよう
- ヒント：「`//`」は整数の割り算で、小数点以下は切り捨て  
(追加問題) 小数点を表示させるにはどうすればいいか？

## 問2

- “apple”という文字列の長さを表示させてみよう
- ヒント：組み込み関数を使う  
(追加問題) 一文字ずつ表示させてみよう
- ヒント：繰り返しを利用すると

# 基礎編問題

## 問3

- ['apple', 'orange', 'banana', 'peach'] というリストの3番目の要素を表示させてみよう
- ヒント：リストの添え字（インデックス）は0から始まる
- （追加問題）一要素ずつ表示させてみよう
- ヒント：繰り返しを利用すると

## 問4

- 'zero,one,two,three' という文字列を、','で区切って、短い文字列のリストを作り、それをそのまま表示させよう
- ヒント：文字列型のメソッドを使う
- （追加問題）リスト化したものを、再び文字列化しよう
- ヒント：文字列 = '区切り文字'.join(リスト)

# 基礎編問題

## 問5

- 変数x の値が負の値であれば‘negative’と、そうでなければ‘nonnegative’ と表示させるif文を書いてみよう
- ヒント：if ~else 文を使う  
(追加問題) xが0のときは“zero”と表示させよう
- ヒント：elifを利用しよう

## 問6

- 11から20までの自然数を、1つずつ、改行しながら表示させるforループを書いてみよう
- ヒント：range関数を使うとよい  
(追加問題) 11から20までの自然数を1つ飛ばしで表示させるforループを書いてみよう
- **range(start,stop,step)**→引数に整数を3つ指定すると、 $\text{start} \leq i < \text{stop}$ でstepずつ増加する等差数列が生成される。

# 基礎編問題

## 問7

- 与えられた数を2倍して返す関数を書き、それに適当な値を渡して呼び出すプログラムを書いてみよう
- ヒント：returnを使って値を返すのを忘れないように

# 標準編問題

## 問1

- 1000以上2000以下の整数のうち、
- 7で割り切れ、かつ、5で割り切れないものをすべて、
- 「,」で区切って画面に表示してみよう。

→ここから具体的なヒントではなく、問題をプログラミングに変換できるように意識しながら考えて行きましょう！

- 1000以上2000以下の整数のうち→for文を使うのか
- 7で割り切れ、かつ、5で割り切れないものをすべて  
→if文を使うのか...どのような比較演算子を使えばよいか？  
→or・and・not演算子を利用できないか？
- 「,」で区切って画面に表示してみよう。  
→リスト化して、その後まとめて文字列化すればいいのか？

# 標準編問題

## 問2

- 「,」で区切られた複数の単語の列を、例：“with,go,apple,late,set”
- 1つの長い文字列として、キーボード入力で受け取り、
- それらの単語を辞書式順に並び替えてから、
- やはり「,」で区切られた単語の列として、
- 画面に表示してみよう。

→複数の単語列を標準入力（コンソールからの入力）できるようにしてみよう

→`input_name = input()`

→これを並び替える・・・文字列の変換が必要

→並び替え、リスト関数の中の何を使えばいいのか？

→もう一度リストを文字列に戻す必要がある

→画面の表示方法は？



# 標準編問題

## 問3

- 「,」で区切られた複数の単語の列を、例：“with,go,apple,late,set”
- 1つの長い文字列として、キーボード入力で受け取り、
- 各々の単語をキー(key)とし、
- その単語の長さを値(value)とするような辞書を作り、
- その辞書をそのまま画面に表示させてみよう

→例えば標準入力を行う

→各単語の長さをどのように習得する？

→リスト化した後に、個々の文字列の長さや文字列を生成する方法は？

→辞書はどのように作るか？

→辞書の更新はどうすればよいか？

# 標準編問題

## 問4

- 数をユーザに当てさせるゲームを作ろう。
- プログラムの初めに0以上99以下の整数の乱数を発生させる。
- この数を、以下のようにしてユーザに当てさせる。
- ユーザは毎回キーボードから数を入力する。
- もしその数が正解なら、正解である旨を表示してプログラムは終了。
- そうでないなら、正解よりも大きな値か、あるいは小さな値かを、ヒントとして画面に表示する。
- 正解にたどりつくまで、ユーザに数を入力させる。

### →乱数の生成方法

```
import random
```

```
correct_number = random.randint(a,b) この生成範囲は (a<= n <= b)  
となる
```

→for以外の繰り返しは？→条件分離付きの繰り返し方法がある！→While文  
→入力する文字は、「文字列型」→比較は「整数型」変化が必要

# 標準編問題

## 問5

- キーボード入力で、
- 「0」と「1」だけから成る文字列を受け取り、
- その文字列を2進数の数と解釈して、
- 同じ値を8進数で表した場合の表記を文字列として作り、
- 画面に表示させてみよう。

→標準入力が必要

→文字列を2進数として考える

(一度文字列を2進数から10進数に変化させると?)

→その2進数として考えた文字列を8進数として変化させる必要がある

→組み込み関数で存在するのか?

# 応用課題

## ■選択ソートもしくはバブルソートを記述してみましょう

```
from random import shuffle  
l = list(range(15))  
print(l)  
shuffle(l)  
n = len(l)
```

ここから書いてみてください・・・！

ソート前の配列



1. 先頭「3」と残りの右側の要素を比較する。 (「1」と交換)



2. 比較元の要素を右に進め「9」と残りの右側の要素を比較する。 (「2」と交換)



3. 比較元の要素を右に進め「6」と残りの右側の要素を比較する。 (「3」と交換)



4. 比較元の要素を右に進め「6」と残りの右側の要素を比較する。 (交換しない)



ソート完了



色の意味

