


The background is a vibrant, abstract composition of light trails and streaks in shades of blue, teal, and yellow, creating a sense of motion and depth. A large, white, rounded rectangular box is centered in the middle of the image, serving as a backdrop for the text.

Python その1

この資料について



オンライン学習プラットフォーム『Udemy』
で公開している、
『Python x FastAPI初心者向け講座』の
説明用資料です

[https://www.udemy.com/course/
python_fastapi](https://www.udemy.com/course/python_fastapi)



本講座の内容

本講座の内容



Python初心者向け

Python基本知識の習得

PythonでAPIサーバをつくれるようになる

Python + FastAPI



Pythonの概要

Pythonの概要



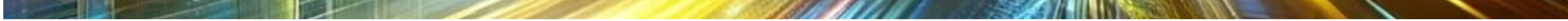
1994年に誕生

オープンソース(無料)のプログラム言語

開発は全世界のボランティア

開発資金は寄付、スポンサーシップなど

Pythonの特徴



1. 用途の広さ

Web開発、データ分析、人工知能(AI)、化学計算, IoTなど

2. シンプルで読みやすい

3. 豊富なライブラリ (追加機能)

フロントエンドとバックエンド



クライアント



サーバー



Pythonの年表



1994年 Python 1.0

2000年10月 Python 2.0

2008年12月 Python 3.0

2014年3月 Python 3.4 pip追加

2021年10月 Python 3.10.0

2023年10月 Python 3.12.0

毎年アップデート中



開発環境

開発環境の種類

	google colaboratory (Jupyter Book)	venv	anaconda (conda)	コンテナ (Docker)
特徴	ブラウザから 直接使える コード、注釈、視覚化を まとめて表示	ローカルマシンにPythonと 仮想環境を設定	科学計算、データサイエンスに 特化した開発環境	ソフトウェアを コンテナとしてパック 異なる環境で開発しやすい
メリット	インストール不要で 手軽に使える	プロジェクト毎に 異なる環境を簡単に設定	科学計算やデータサイエンス向 けのパッケージが豊富 JupyterBook込み GUIでパッケージ管理できる	環境の再現性が高い データベースもコンテナ化し て使える
デメリット	インターネット接続必須 大規模開発には不向き	依存関係の管理に注意が必要	インストールサイズが大きい 必要ないパッケージも 多く含まれることがある	設定が複雑になりがち コンテナ技術の知識が必要

Python 開発環境

win

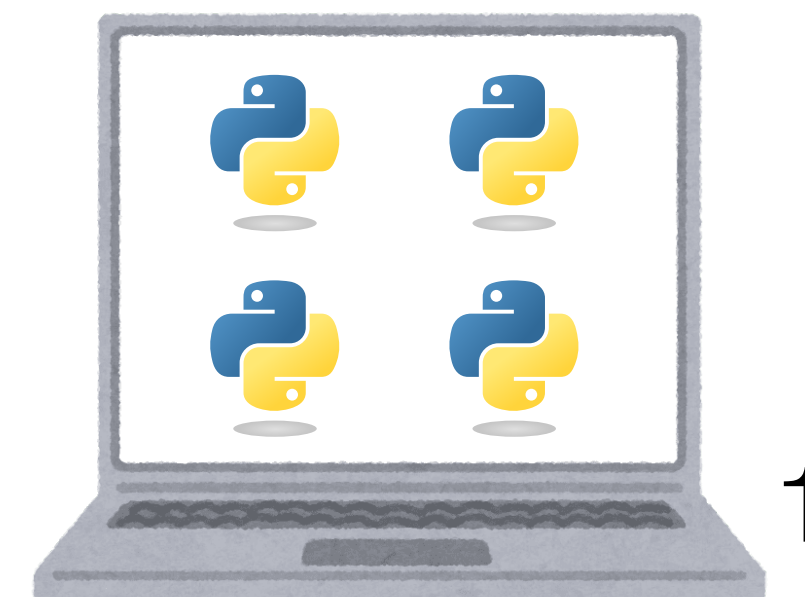
Pythonランチャー ・ ・ 複数のPythonバージョン管理

venv ・ ・ Pythonの仮想環境を作成 ・ 管理

mac

pyenv ・ ・ ・ ・ 複数のPythonバージョン管理

venv ・ ・ Pythonの仮想環境を作成 ・ 管理



Pythonのインストール(mac) 1



macOS環境のPython

<https://www.python.jp/install/macos/index.html>

1. homebrewインストール <https://brew.sh/ja/>

2. パスを追加

echo \$SHELL

zshなら ~/.zshrc

bashなら ~/.bash_profile

viエディタかメモ帳 viエディタなら iで編集モード Escで閲覧モード :wqで上書き保存

FinderでShift + Cmd + . で隠しファイル表示

Pythonのインストール(mac) 2




```
export PATH="$PATH:/opt/homebrew/bin/"  
export PYENV_ROOT="$HOME/.pyenv"  
export PATH="$PYENV_ROOT/shims:$PATH"  
eval "$(pyenv init -)"
```

source ~/.ファイル名 # 設定を読み込ませる

brew -v でバージョン表示されればOK

Pythonのインストール(mac) 3



3. pyenvインストール

```
brew install pyenv
```

```
pyenv install --list # インストールできるpythonバージョン確認
```

```
pyenv install 3.12.0 # インストール
```

```
pyenv versions # インストールバージョン確認
```

4. pythonバージョン選択

```
pyenv global 3.12.0 # globalで設定
```

```
python3 --version
```


Python実行



シンボリックリンクの作成

```
echo 'export PATH="$(brew --prefix python)/libexec/bin:  
$PATH"' >> ~/.zshrc
```

```
source ~/.zshrc
```

python -V でバージョン表示されればok

Python リンク



Python公式ページ

<https://www.python.org/>

Pythonドキュメント

<https://docs.python.org/ja/3/>

Python japan(環境構築ガイド)

<https://www.python.jp/install/install.html>

Pythonのインストール(win)



パッケージダウンロード

<https://www.python.org/downloads/> (公式・わかりにくい)

<https://pythonlinks.python.jp/ja/index.html>

Add Python 3.x to PATH をチェック

インストールを進める

コマンドプロンプト/ Powershellを開き

python -V と入力しバージョン表示されればok

仮想環境の作成その1



任意のフォルダで仮想環境作成

mac

pwd で現在フォルダ確認

/Users/{ユーザー名}/python/python-test

win

C:¥python¥python-test

仮想環境の作成 その2



仮想環境の作成 (-mはモジュール名指定)

```
$ python -m venv .venv
```

有効化

```
$ . .venv/bin/activate
```

```
(.venv) $
```

終了

```
(.venv) $ deactivate
```

```
$
```




VSCode

VSCode拡張機能



VSCode <https://azure.microsoft.com/ja-jp/products/visual-studio-code>

Python 必須

Pylance インテリセンス(コード補完)やエラー表示

Python Type Hint 型のヒント

indent-rainbow

下記はお好みで

Dracula Theme

Material Icon Theme



Pythonの書き方 実行方法

Pythonの書き方・実行方法

venvを有効化した状態で進めます

文字列以外は全て半角

ファイル拡張子は xxx.py

(.venv) \$ section2/**first.py**

```
print(123)
```

```
print("あいうえお") # 文字列はシングルかダブルコーテーションで囲む
```

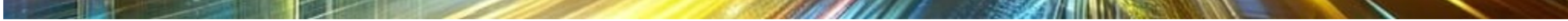
```
# コメントです
```

(.venv) \$ python section2/first.py # 実行



変数

変数



箱のようなもの

箱の名前 = 中身の情報

Pythonは動的プログラミング言語(型を自動判定)

variable.py

```
test_number = 123 #数字
```

```
test_string = "あいうえお" #文字列
```

```
test_bool = True #ブール型(True, False)
```

```
test_none = None #None型
```

```
print(test_number, test_string, test_bool, test_none)
```

変数の命名規則



全て半角

先頭は文字か_

大文字小文字は区別される

できるだけ意味のわかる名詞・英語で

2単語以上はスネークケース推奨 test_name

PEP8(コーディング規約) 参照

<https://pep8-jp.readthedocs.io/ja/latest/>

定数



Pythonは厳密な定数はない

慣習として全て大文字とアンダーバーで定義する

TAX = 0.1



コレクション データ型

コレクションデータ型



リスト(list) ・ ・ 配列 [1, 2, 3]

辞書(dict) ・ ・ 連想配列 {"key" : "value" }

タプル(tuple) ・ ・ 変更不可 (1, 2, 3)

セット(set) ・ ・ 重複しない {1, 2, 3}

リスト(配列) 1行



リストの指定

```
list_1 = ["あああ", 2, 3]
```

リスト内指定

0からスタート

```
print(list_1[1])
```


リスト (配列) 2行・3列



横が行
縦が列

リスト (配列) 3行・4列



仕組みは同じ

リスト コード

list.py

リスト 1行

```
list_1 = ["あああ", 2, 3]
```

```
print(list_1[1])
```

リスト 2行2列

```
list_2 = [  
    ["赤","青","黄"],  
    ["緑","紫","黒"]  
]
```

```
print(list_2[0][1])
```

リスト型と辞書型の違い



リスト型 ・ ・ 数字(順番固定)と値がセット。

`list_1[1]`

辞書型 ・ ・ キーと値がセット。

キー：値

`player["name"]`

辞書型

名前と値がセット (キー: バリュー)

dict.py

```
player = {  
    "name": "三苫",  
    "height": 170,  
    "hobby": "サッカー"  
}
```

```
print(player) # 辞書型の内容全て表示
```

```
print(player["name"]) # nameの情報のみ表示
```

辞書型 (学校の例)



辞書型 (学校の例)

1組



0:本田



1:香川



2:長友



3:乾



4:大迫

2組



0:川島



1:柴崎



2:槇野



3:長谷部



4:酒井

辞書型 (学校の例)

2年

1組



0:遠藤



1:内田



2:松井



3:岡崎



4:中村

2組



0:今野



1:駒野



2:大久保



3:楢崎



4:闘莉王

1年

1組



0:本田



1:香川



2:長友



3:乾



4:大迫

2組



0:川島



1:柴崎



2:槇野



3:長谷部



4:酒井

辞書型 (学校の例)

福岡小学校

2年

1組

2組

熊本小学校

2年

1組



0:遠藤



1:内田



2:松井



3:岡崎



4:中村

2組



0:今野



1:駒野



2:大久保



3:樫崎



4:関莉王

1年

1組



0:本田



1:香川



2:長友



3:乾



4:大迫

2組



0:川島



1:柴崎



2:横野



3:長谷部



4:酒井

多次元(多段) 辞書型

dict2.py

```
students = {  
    "classA" : [  
        {"name" : "三苫", "height" : 170},  
        {"name" : "伊東", "height" : 165},  
        {"name" : "久保", "height" : 168},  
    ],  
    "classB" : [  
        {"name" : "遠藤", "height" : 175},  
        {"name" : "南野", "height" : 163},  
        {"name" : "田中", "height" : 162},  
    ]  
}
```

```
print(students["classA"][1]["name"]) # 伊東が表示される
```


タプル(tuple)

変更できないので定数のように使うこともある

タプル

```
BLOOD_TUPLE = ("A", "B", "AB", "O")
```

```
# BLOOD_TUPLE[2] = "AB+" # 書き換え不可
```

```
print(BLOOD_TUPLE)
```

リスト

```
blood_list = ["A", "B", "AB", "O"]
```


```
blood_list[2] = "AB+"
```

```
print(blood_list)
```




演算子

演算子(計算や判定) 抜粋



四則演算子 $+$, $-$, $*$, $/$, $\%$

比較演算子 $>$, $>=$, $+=$, $==$, $!=$

論理演算子 `and`, `or`, `not`,

その他 `is`, `is not`

文字列結合 `"文字列" + "文字列"`

演算子の例



enzanshi.py

```
number1 = 8
```

```
number2 = 3
```

```
number3 = number1 % number2
```

```
print(number3) # 余りの2
```

```
string1 = "あいう" + "えお" # 文字の結合
```

```
print("かきく" + "けこ" + string1)
```




条件分歧

条件分岐(if文)



Pythonはインデント(段落)が重要

(インデントを強制 カッコはない

コードが見やすい、誰が書いても同じフォーマット)

タブキーではなく半角スペース推奨(4文字分)

if(条件式):

真なら実行

if文 例その1

if.py

```
height = 167
```

```
if(height >= 170):
```

```
    print("身長は170cm以上")
```

```
if(height >= 160 and height < 170):
```

```
    print("身長は160cm以上かつ170cm未満")
```

andを省略して書く事もできる

```
if(160 <= height < 170):
```

```
    print("身長は160cm～170cm未満")
```

if文 例その2



if.py

```
signal = "blue"
```

```
if(signal == "blue"):
```

```
    print("青")
```

```
elif(signal == "yellow"):
```

```
    print("黄")
```

```
else:
```

```
    print("赤")
```


if文応用



if **not** user: # ユーザーがいなかったら

if User **is** None: # ユーザーがNoneなら



繰り返し

繰り返し (for文) 回数指定

whileもあるがPythonの場合 for で事足りる

for.py

繰り返す回数を指定

#for 変数 in 繰り返す対象:

range() ・ ・ 指定された範囲の整数を生成

for i **in** range(1, 6):

 print(i)

繰り返し (for文) 全件表示



リスト全件表示

for.py

```
items = ["apple", "banana", "berry"]
```

for 単数系 in 複数形:

```
for item in items: # 1件ずつ取り出す
```

```
    print(item)
```


繰り返し(for文) 多次元リスト(辞書型) その1

多次元の場合はfor文を組み合わせる

```
students = {略}
```

```
# items() 辞書型向けの関数(メソッド)
```

```
# 辞書のキーと値のペアをタプルで返す
```

```
for class_name, students_list in students.items():  
    print(class_name, students_list)
```

繰り返し(for文) 多次元リスト(辞書型) その2

students_listがリスト型なので再度for文を使う

print内で変数を表示するには

f-strings(フォーマット済み文字列リテラル)を使う

for2.py

```
for class_name, students_list in students.items():
```

```
    # print(class_name, students_list)
```

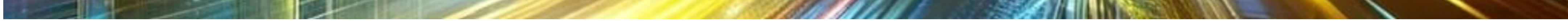
```
    for student in students_list:
```

```
        print(f"名前: {student['name']}, 身長: {student['height']}")
```




内包表記

内包表記 *慣れてからでok



if文・for文を1行で書ける

naiho.py

for文

変数 = [処理 for 変数名 in 対象]

double = [i * 2 for i in range(5)]

print(double)

内包表記 *慣れてからでok

naiho.py

if文

[処理 for 変数名 in 対象 if 条件式]

```
odds = [ i for i in range(10) if i % 2 == 1 ]
```

```
print(odds)
```

elseも組み合わせられるけれど複雑になりがち



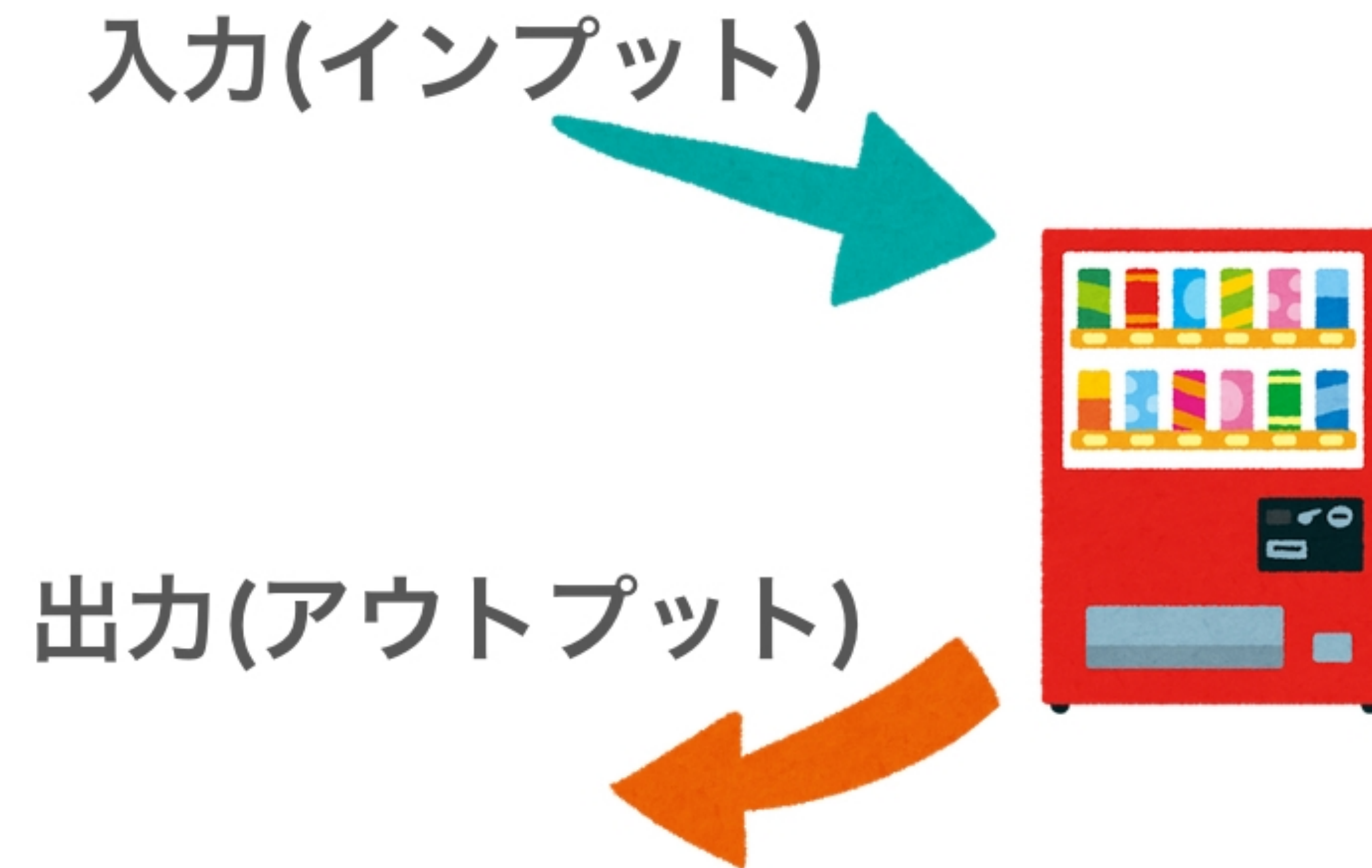
関数

関数 ・ ・ function



関数 ・ ・ function ・ ・ 機能

関数・・何らかの処理



数学の関数と似ている

入力(インプット)



出力(アウトプット)

$$f(x) = 5 * x + 10$$

関数は2種類



組み込み関数(ビルトイン関数)

- ・ 準備してある関数

ユーザー定義関数

- ・ 自由に作れる関数

関数の作り方(基本)

関数の定義

```
def 関数名(仮の引数):
```

```
    ～処理の内容～
```

```
    return 戻り値
```

関数を使う時

関数名(実際の引数)

関数名は名詞 または 動詞+名詞 名前だけで意味がわかると良

関数の例 1

function.py

インプット引数なし、アウトプット戻り値 なし

```
def test():
```

```
    print("テスト")
```

```
test()
```

引数あり、戻り値なし

```
comment = "コメント"
```

```
def get_comment(string):
```

```
    print(string)
```

```
get_comment(comment)
```


関数の例 2

引数なし、戻り値あり

```
def get_number_of_comment():  
    return 5
```

```
comment_number = get_number_of_comment() # 関数の戻り値を変数に入れている  
print(comment_number)
```

引数2つ、戻り値あり

```
def sum_price(int1, int2):  
    int3 = int1 + int2  
    return int3
```

```
total = sum_price(3, 5)  
print(total)
```

よく使うビルトイン関数

print() テキスト出力 len() 要素の数 type() オブジェクトの型

int(), str() 整数、文字列への型変換

range() 連続する数値を生成

max(), min() 最大値、最小値 sum() 合計

built_in.py

```
print(len("あいうえお")) # 5
```

```
print(type("文字")) # <class 'str'>
```

```
print(max([1, 2, 3])) # 3
```

```
print(sum([1, 2, 3])) # 6
```




スコープ

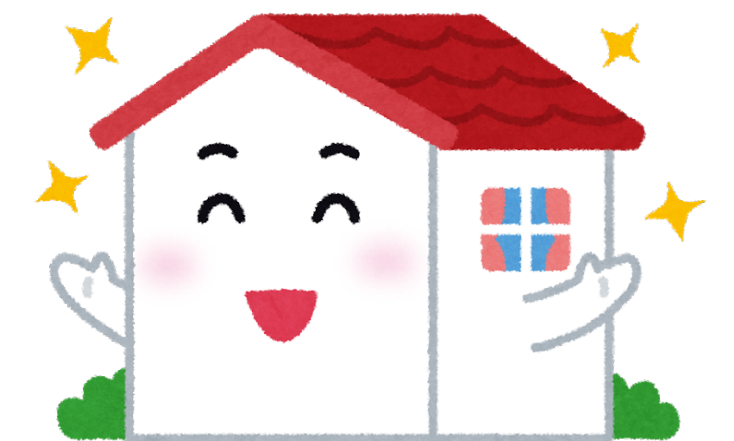
スコープ (範囲)

グローバルスコープ・・・世界



関数の外側で変数を設定、どこでも使える

ローカルスコープ・・・それぞれ



関数の内側で変数を設定、関数の中でしか使えない

スコープ

scope.py

```
global_variable = "グローバル"
```

```
def global_test():  
    print(global_variable)
```

```
def local_test():  
    local_variable = "ローカル"  
    print(local_variable)
```

```
global_test()
```

```
local_test()
```

```
print(local_variable) # エラー発生
```




メソッド

メソッド



関数のようなもの

後ほどクラスと合わせて再紹介します

リスト、タプル、辞書、文字列

それぞれに使える専用の関数(メソッド)

変数名.`メソッド名()`

メソッド 抜粋

メソッド	機能	リスト	タプル	辞書	文字列
.append()	リストの末尾に要素を追加する	○			
.extend()	リストの末尾にリスト、タプル、辞書を追加できる	○			
.count()	ある要素がリストにいくつ含まれるか数える	○	○		
.pop()	指定したインデックスの要素を1つだけ抜き出す	○		○	
.keys()	辞書型のkeyについてループ処理する			○	
.values()	辞書型のvalueについてループ処理する			○	
.items()	辞書型のkeyとvalueをループ処理する			○	
.replace(a, b)	文字列aをbに置き換える				○
.split()	文字列を特定の文字で分割する				○
.join()	連結した文字列を得る				○
.index()	最初に指定した文字が現れる位置を返す				○

メソッド 例 1

method.py

リスト

```
list_1 = [1, 2, 3]
```

```
list_1.append(4) # 追加
```

```
print(list_1) # [1, 2, 3, 4]
```

```
my_list = [1, 2, 3]
```

```
another_list = [4, 5, 6]
```

```
my_list.extend(another_list) # 別のリストを追加
```

```
print(my_list) # [1, 2, 3, 4, 5, 6]
```

タプル

```
my_tuple = (1, 2, 2, 3, 4)
```

```
count = my_tuple.count(2) # 2の数を調べる
```

```
print(count) # 2
```

メソッド 例 2

辞書

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
```

```
values = my_dict.values() # valuesを取得
```

```
print(list(values)) # [1, 2, 3]
```

文字列

```
text = "hello world"
```

```
new_text = text.replace("world", "Python") # 置換
```

```
print(new_text) # hello Python
```

```
text = "one, two, three"
```

```
words = text.split(", ") # 分割
```

```
print(words) # ['one', 'two', 'three']
```

```
text = "hello world"
```

```
index = text.index("world")
```

```
print(index) # 6
```