

# Python活用基礎研修

## 入門編1: Pythonプログラミングの基本

# Python活用基礎研修について

## 研修の目標

- Pythonを通してプログラミングの基礎を学ぶ.
- 実際に手を動かして簡単なデータ解析およびAI（機械学習）を体験する.

## 入門編

- 入門編1: Pythonプログラミングの基本的なルールを学ぶ.
- 入門編2: NumPyを用いたプログラミングを学ぶ.

## 応用編: 各種モジュール（=便利なツール）の使い方を学ぶ

- 応用編1（データ解析）: pandas, Matplotlib
- 応用編2（AI, 機械学習）: scikit-learn

## 入門編1: Pythonプログラミングの基本

- 0. Pythonとは？
- 1. Pythonを使ってみる
- 2. Jupyter Notebookの使い方
- 3. プログラミング入門
- 4. データ構造: リスト

# 0. Pythonとは

**Python**は、1991年にグイド・ヴァン・ロッサムによって開発されたプログラミング言語。高い可読性や実用的なモジュールが充実している点から近年広く利用されている。プログラミング教育において、C言語に代わりPythonが使われることが増えている。

## 身近なPythonの使用例

- Webサービス: YouTube, Instagram, Dropbox, Spotify

## Pythonの実用的なモジュール（=ツール）の例

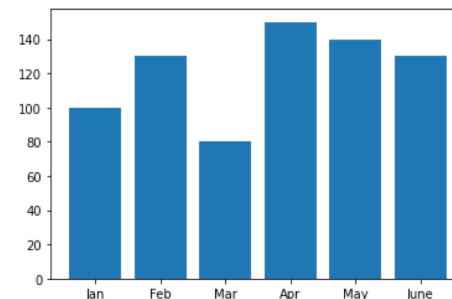
- 自動化: Beautiful Soup, Scrapy
- 科学技術計算: NumPy, SciPy, SymPy
- データ解析: pandas, Matplotlib
- AI(機械学習): scikit-learn, TensorFlow, Keras, PyTorch

# 1. Pythonを使ってみる

## 簡単な計算

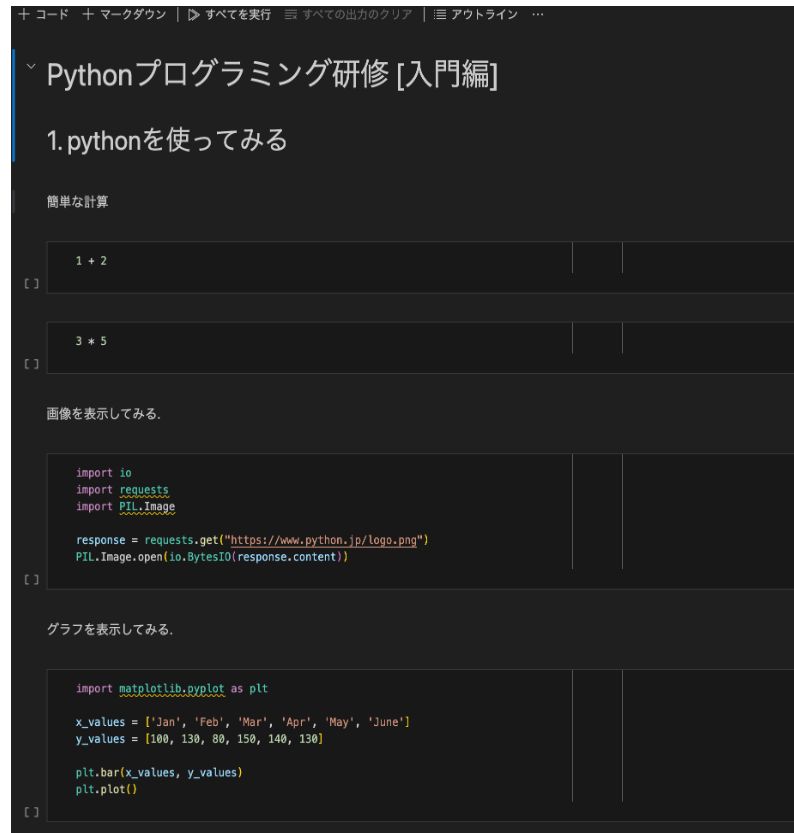
```
1 + 2  
3 * 5
```

## 画像やグラフの表示



# 2.1 Jupyter Notebookとは？

**Jupyter Notebook**とは、ブラウザ上でPythonを実行できる対話型の実行環境。  
実行結果を逐次確認しながらプログラミングできる。



The screenshot shows a Jupyter Notebook interface with a dark theme. At the top, there's a toolbar with icons for code, markdown, and execution. Below the toolbar, the notebook title is "Pythonプログラミング研修 [入門編]". The first section is "1. pythonを使ってみる". Under this section, there are two code cells. The first code cell contains the text "簡単な計算" followed by a code block with "1 + 2". The second code cell contains the text "3 \* 5". Below these, there's a section titled "画像を表示してみる." followed by a code cell containing Python code to fetch and display the Python logo. The final section is "グラフを表示してみる." followed by a code cell containing Python code to create a bar chart using matplotlib.

```
Pythonプログラミング研修 [入門編]

1. pythonを使ってみる

簡単な計算

1 + 2

3 * 5

画像を表示してみる.

import io
import requests
import PIL.Image

response = requests.get("https://www.python.jp/logo.png")
PIL.Image.open(io.BytesIO(response.content))

グラフを表示してみる.

import matplotlib.pyplot as plt

x_values = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June']
y_values = [100, 130, 80, 150, 140, 130]

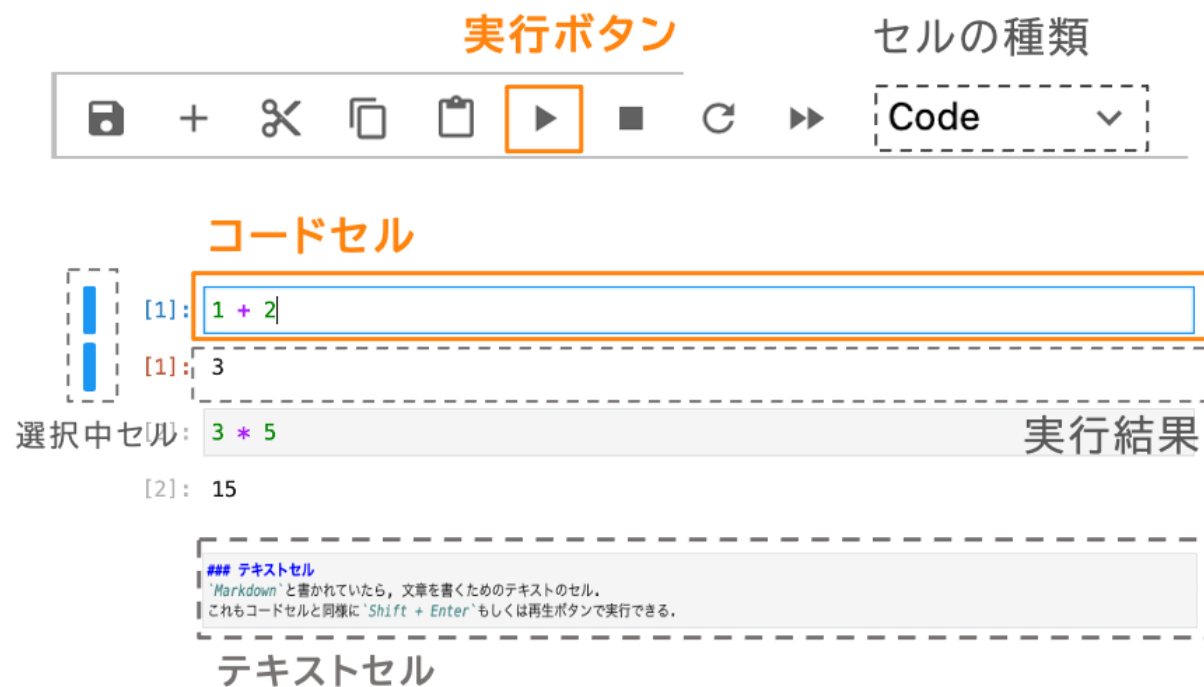
plt.bar(x_values, y_values)
plt.plot()
```

## 2.2 操作方法: プログラムの実行

コードセルという四角の領域に実行したいコードを入力する.

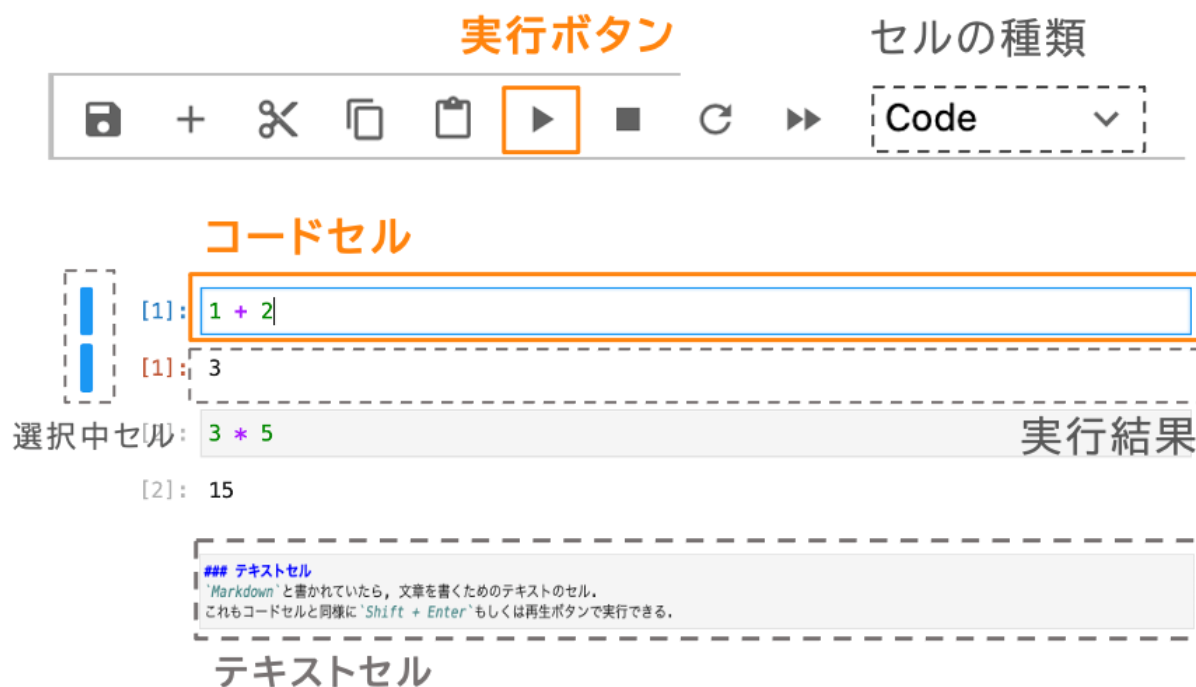
その後、`Shift + Enter` もしくは実行ボタンを押す.

テキストセルは説明やメモなどを記述可能.



## 2.2 操作方法: セルの移動とセルの変更

- セルの移動: セルをクリック, もしくは上▲キーまたは下▼キーを押下
- セルの追加: Bキーもしくはセル内の Insert a cell below をクリック
- セルの種類をコードセルへ変更: セルを選択し Yキーを押下
- セルの種類をテキストセルへ変更: セルを選択し Mキーを押下



# 入門編1

1. Pythonを使ってみる
2. Jupyter Notebookの使い方
3. **プログラミング入門**
  - 3.1 コメントとコメントアウト
  - 3.2 数値
  - 3.3 文字列
  - 3.4 変数
  - 3.5 関数
  - 3.6 モジュールとimport
  - 3.7 分岐処理
  - 3.8 練習問題
4. データ構造: リスト

## 3.1 コメントとコメントアウト

コメントは, プログラムの内容について記述する (=コメント) 場合や  
プログラムの一部を無効にする (=コメントアウト) 場合に用いる.  
具体的には, # から行末までの範囲はプログラム実行時に無視される.

```
1 + 2 # ここはコメント
```

```
# この行は無視
```

```
5 + 7
```

```
# 以下の計算は実行されない (=コメントアウト)
```

```
# 10 - 8
```

## 3.2 数値

整数同士の四則演算において計算結果は以下の通りである.

- 足し算 `+`, 引き算 `-`, および, かけ算 `*` の計算結果は整数 `int`型
- 割り算 `/` の計算結果は実数 `float`型

```
1 + 2 # 計算結果は, 3 (int型)
1 - 5 # 計算結果は, -4 (int型)
30 * 2 # 計算結果は, 60 (int型)
2 / 3 # 計算結果は, 0.6666666666666666 (float型)
```

## 3.3 文字列

文字列をプログラム内で使用するときは、  
文字列の前後を `"` (ダブルクォーテーション) または `'` (シングルクォーテーション) で囲む。

```
"Python Programming" # ダブルクォーテーションで文字列を定義する場合  
'Python Programming' # シングルクォーテーションで文字列を定義する場合  
"技術職員研修"       # 文字列として日本語も使用可能
```

### 数値と文字列の違い

数値同士は計算できるが、文字列同士は計算できない。  
(では、数値と文字列の計算は実行できるのか?)

```
123 * 456      # 計算結果として56088が表示される  
'123' * '456' # エラー
```

## 3.4 変数: 定義

変数を用いることで、計算結果の一時的な保存や入力した値の再利用が実現できる。具体的には、数字や文字列などを名前のついた入れ物（=変数）に格納できる。コード上の文字列と変数を区別する唯一の鍵は `"` または `'` で囲われているかどうか。

```
apple = 150    # 変数appleに数値150を代入
'apple' = 150  # 文字列appleは変数ではないため、数値150を代入できないためエラー
apple + 300    # 変数appleは、これまでの数値と同様に計算が可能
```

変数には、int型やfloat型以外の型も代入可能。

```
orange = 'オレンジ' # 文字列型
prices = [300, 150, 400] # 後に説明するリスト(配列)
prices
```

## 3.4 変数: 値の確認 (print関数の使用)

`print` 関数を使うことで変数に代入された値や文字列や数値などを画面に出力できる。  
ただしJupyter Notebook上では`print`関数を用いなくても出力が可能。

```
a = 123
print(a)          # 数値を出力
print("Python")   # 文字列を出力
'abc'             # Jupyter Notebookなら左のように書くだけで出力可能。
```

## 3.5 関数

**関数**とは, 与えられた入力 (= 引数) を元に何かしらの処理を実行し, その結果を出力 (= 戻り値) として返すもの.

以下の書き方で関数を使用できる.

```
関数名(引数1, 引数2, ...)  
変数 = 関数名(引数1, 引数2, ...) # 左のように書くと戻り値を変数に代入できる
```

- 値を表示する関数 `print`
- 絶対値を求める関数 `abs`
- 最大値を求める関数 `max`

```
print("Python") # 引数は文字列Python, 戻り値はなし  
abs(-30)        # 引数は数値-30, 戻り値は数値30  
max(1, -3, 5, 2) # 引数は数値1, 数値-3, 数値5, 数値2, 戻り値は数値5
```

## 3.6 モジュールとimport

**モジュール**はPythonのさまざまな関数を目的や種類ごとにまとめたもの。

例えば、三角関数などの数学関係の機能は `math` モジュールにまとめられている。

モジュールの関数を使用したい場合、必ず `import` でモジュールを指定する必要がある。

その後、`モジュール名.関数名()` でモジュール内の関数を使うことができる。

- `math` モジュール内の関数 `sqrt` を用いて平方根を求める場合。

```
import math
math.sqrt(3) # 関数sqrtに数値3を入力し、その計算計算結果が出力される
```

- 上記から引き続き、`math` モジュール内の関数 `sin` でsinを計算する場合。

```
pi = 3.141592653589793 # math.piと書くことでmathで定義された円周率の変数piも使用可能
math.sin(pi/2)         # 関数sinに数値  $\pi/2$ を入力し、その計算計算結果が出力される
```

## 3.7 分岐処理: if文

if 文は, 条件に応じて処理する内容を分岐したい場合に用いる.

if 文の右側に記述された **条件式** が満たされれば直下の処理が実行される.

```
a = 85
```

```
if a > 60:           # 条件式: 変数aが数値60を超えてるかどうか  
    print('合格です!') # 条件式が満たされた場合の処理
```

主な条件式

演算子	条件
$a < b$	a は b より小さい
$a \leq b$	a は b と等しいか小さい
$a > b$	a は b より大きい
$a \geq b$	a は b と等しいか大きい
$a == b$	a と b は等しい
$a != b$	a と b は等しくない

# 余談: インデント

**インデント (=字下げ)** を用いることで, if文などの処理が及ぶ範囲を表現する.  
インデントは文頭にスペースまたはタブを1つ以上記述することで表現される.  
(コードの可読性の観点からスペース4つを使用することが推奨されている.)

```
a = 85
if a > 60:                # 条件式: 変数aが数値60を超えてるかどうか
    print('合格です!')    # 条件式が満たされた場合の処理
print('通知内容は以上の通りです') # if文の条件文に関係なく実行される
```

上記の例では, 1つめのprint関数を記述する際にインデントを行なっている.  
一方で, 2つめのprint関数はインデントされていないため,  
if 文の条件文に関係なく実行される.

## 3.7 分岐処理: else節とelif節

else 節を用いることで if 文の条件式が満たされなかった場合の処理を指定できる。さらに、複数の条件を指定する場合は elif 節を用いる。

```
a = 55
if a > 60:          # 条件式: 変数aが数値60を超えてるかどう
    print('合格です!') # 条件式が満たされた場合の処理
else:
    print('不合格です.') # 条件式が満たされない場合の処理

a = 65
if a >= 80:          # 条件式1: 変数aが数値60を超えてるかどう
    print('A判定で合格です!') # ifの条件式1が満たされた場合の処理
elif a >= 60 and a < 80: # 条件式2: 変数aが数値60から数値80に収まるかどうか
    print('B判定で合格です!') # elifの条件式2が満たされた場合の処理
else:
    print('不合格です.') # 全ての条件式が満たされない場合の処理
```

## 3.8 練習問題: その1

下記のプログラム上の `a` の値を変更して, 各条件式の処理を実行せよ.

```
a = 65
if a >= 80:                # 条件式1: 変数aが数値60を超えてるかどうか
    print('A判定で合格です!') # ifの条件式1が満たされた場合の処理
elif a >= 60 and a < 80:    # 条件式2: 変数aが数値60から数値80に収まるかどうか
    print('B判定で合格です!') # elifの条件式2が満たされた場合の処理
else:
    print('不合格です.')    # 全ての条件式が満たされない場合の処理
```

## 3.8 練習問題: その2

以下の表（主な条件式）を参考に自由に条件式を変更し動作を確認せよ.

主な条件式

演算子	条件
$a < b$	a は b より小さい
$a \leq b$	a は b と等しいか小さい
$a > b$	a は b より大きい
$a \geq b$	a は b と等しいか大きい
$a == b$	a と b は等しい
$a != b$	a と b は等しくない

# 入門編1

1. Pythonを使ってみる
2. Jupyterの使い方
3. プログラミング入門
4. **データ構造: リスト**
  - 4.1 リストの定義
  - 4.2 リストの出力
  - 4.3 要素の参照
  - 4.4 要素の置換
  - 4.5 反復処理
  - 4.6 練習問題

# 4.1 リストの定義

リストは、複数の値や文字列をまとめて管理できるデータ構造。

具体的には、下記のように数値や文字列などを `[]` の中に複数並べてることで定義する。

```
a = [1, 3, 5, 7, 9] # 数値のリスト
program = ['Python', 'C', 'Java', 'PHP'] # 文字列のリスト
```

インデックス

0	1	2	3	4
1	3	5	7	9
a[0]	a[1]	a[2]	a[3]	a[4]

リスト **a**

インデックス

0	1	2	3
'Python'	'C'	'Java'	'PHP'
program[0]	program[1]	program[2]	program[3]

リスト **program**

## 4.2 リストの出力

リストが管理する全ての値は `print` , もしくは, Jupyter上であれば `リスト名` で出力できる.

```
print(a)
```

program # Jupyter Notebookなら左のように書くだけで出力可能.

インデックス

0	1	2	3	4
1	3	5	7	9
a[0]	a[1]	a[2]	a[3]	a[4]

リスト `a`

インデックス

0	1	2	3
'Python'	'C'	'Java'	'PHP'
program[0]	program[1]	program[2]	program[3]

リスト `program`

## 4.3 要素の参照

リストの各要素を参照したい場合、**リスト名[要素の番号]**と記述する。  
また、この要素の番号のことを**インデックス**もしくは**添字**という。

インデックスは**0**からはじまることに注意。

なお、インデックスは要素数を超えるものを指定できない。

`a[0]` # リストaの1つめの要素は、添字0を指定することで格納された値1を参照可能  
`a[2]` # リストaの3つめの要素は、添字2を指定することで格納された値5を参照可能

インデックス

0	1	2	3	4
1	3	5	7	9
a[0]	a[1]	a[2]	a[3]	a[4]

リスト **a**

## 4.4 要素の置換

要素の値を置換したい場合, `リスト名[インデックス] = 値` と記述する.

```
program[1] = 'Python' # リストprogramの2つめの要素である文字列'C'を文字列'Python'に置換  
program[4] = 'Perl'   # インデックスは3までなので置換できずエラーとなる
```

0	1	2	3
'Python'	'C'	'Java'	'PHP'
program[0]	program[1]	program[2]	program[0]

リスト **program**

## 4.5 反復処理: for文

for 文を用いることで反復処理を実行できる.  
(ここではリストを用いたfor文のみを扱う.)

```
for 変数名 in リスト名:  
    処理1 # リストの要素を用いた処理などを記述  
    処理2  
    ...
```

リスト `program` の中身を一つずつ繰り返し表示する場合, 以下のように記述する.

```
for s in program:  
    print(s)
```

## 4.6 練習問題: その1

リスト `a` の要素の合計値を `for` 文を用いて計算せよ.

```
a = [1, 3, 5, 7, 9] # 数値のリスト
```

### ヒント

- 合計値を管理するための変数 `goukei` を用意する.
- `goukei = goukei + i` と書くことで,  
右辺の変数 `goukei` の値に変数 `i` の値を加算した計算結果が  
左辺の変数 `goukei` に新たな値として格納される.

## 4.6 練習問題: その2

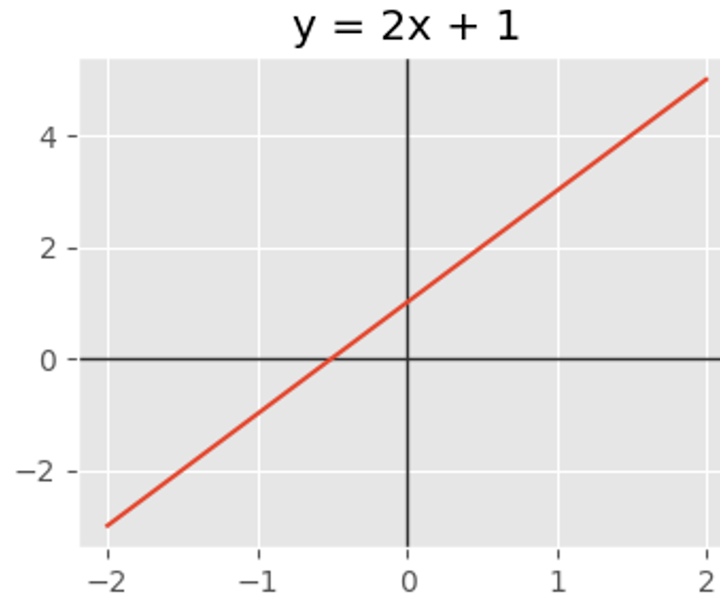
以下の成績表について

1. 山田の成績のリストを作り, 3科目の合計点を求めよ.
2. 数学の成績のリストを作り, 数学の平均点を求めよ.

	名前	国語	数学	英語
0	山田	80	90	75
1	田中	50	95	67
2	鈴木	73	56	95

## 4.6 練習問題: その3

一次方程式  $y = 2x + 1$  があるとき,  $x = [-2, -1, 0, 1, 2]$  のリストを作成し, `for` 文を用いて,  $x$  の各値に対応する  $y$  の値を求めよ.



### ヒント

$y$  の値を管理するためのリスト `y = []` を用意する.

`y.append(a)` と記述することで, リスト `y` に要素 `a` を追加することができる.