

データサイエンティストの道具箱： Pythonによる基礎から実践、そして未来へ

データ処理・分析に不可欠な基本操作と、LLMを活用した最新のワークフローを習得する

なぜ、データサイエンティストにプログラミングが必要なのか？

- データサイエンスのタスク（収集、前処理、分析、可視化）は、手作業では非効率で、再現性も低い。
- プログラミングは、これらの一連の作業を自動化し、効率的に実行するための強力な手段です。
- 特に、機械学習分野ではPythonが標準言語として広く利用されています。

統計学的な処理ではRという言語が使われることも多い。



思考の素材：すべての基本となる「変数」と「型」

左カラム (Definitions)

変数 (Variable)

プログラミングにおいて、値を保持しておくための「名前付きの箱」です。

型 (Type)

その箱に何を入れることができるかを示す「種類」です。



右カラム (Key Examples)

整数型 (int)

1, 57, -100



浮動小数点型 (float)

3.14, 2.5



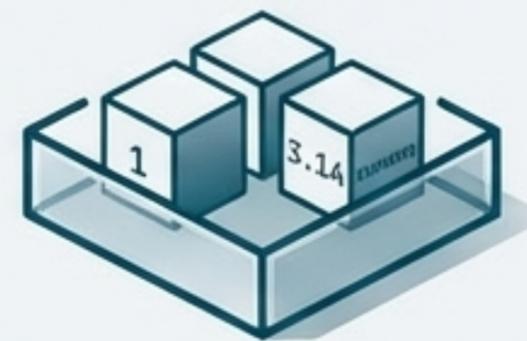
文字列型 (str)

"apple", "Hello World!"



配列型 (list)

[1, 3.14, "banana"]



最初のステップ：変数に値を入れ、計算する

値の代入 (Assignment)

```
# 変数xに1を、変数fruitに"apple"を代入  
x = 1  
fruit = "apple"
```

注意：「=」は「等しい」という意味ではなく、「右辺の値を左辺の変数に代入する」という命令です。

基本的な計算 (Basic Calculations)

```
a = 5  
b = 2  
print(a + b, a - b, a * b, a / b, a // b, a % b)
```

```
7 3 10 2.5 2 1
```

文字列は"または'で囲みます。xは変数、"x"は文字列として解釈されます。

演算子	意味
$a + b$	和
$a - b$	差
$a * b$	積
a / b	(実数としての)商
$a // b$	(a / b)を超えない最大の整数
$a \% b$	$a - b * (a // b)$

$a ** b$ は「aのb乗」を意味します。

ロジックの構築①：条件によって処理を分ける

if文を使って、特定の条件が満たされた場合にのみ、特定の処理を実行できます。

「変数aの値が3の倍数かどうか」を判定する

```
a = 57

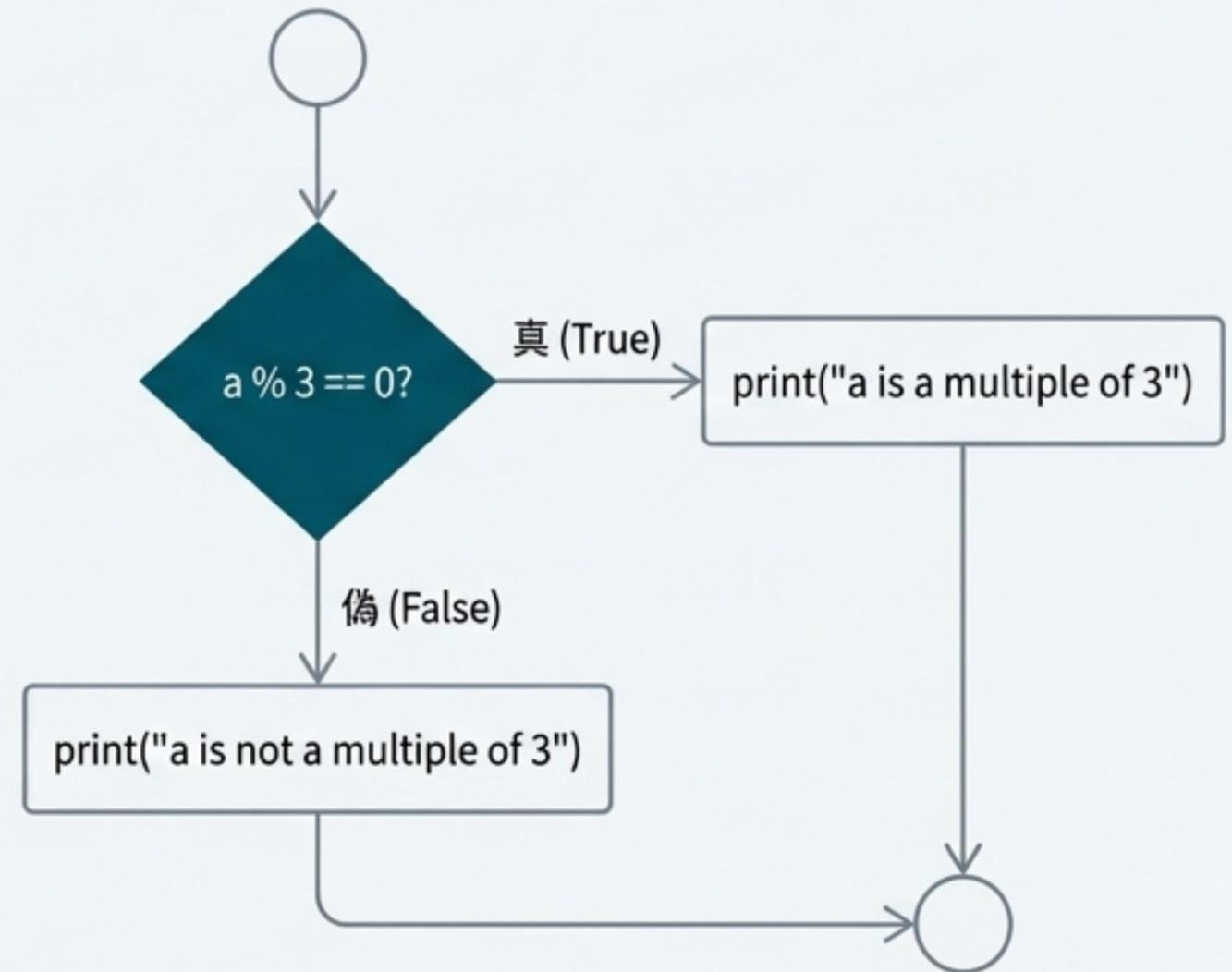
if a % 3 == 0:
    print("a is a multiple of 3")
else:
    print("a is not a multiple of 3")
```

Execution Result

`a is a multiple of 3`

Syntax Breakdown

1. `if a % 3 == 0:` 条件式。「aを3で割った余りが0に等しい」か？
2. `print(...):` 条件が真 (True) の場合の処理。インデントを下げて記述します。
3. `else:` 条件が偽 (False) の場合の処理を記述します。
4. `elif` (条件B): `else if` の略。別の条件でさらに分岐させたい場合に使用します。



最も重要な注意点：代入 `=` と比較 `==`

この2つは全く異なる意味を持ちます。混同はバグの最も一般的な原因の一つです。

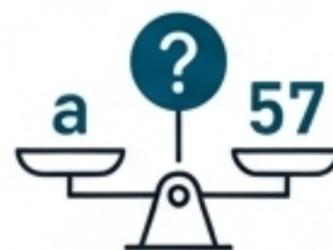
`a = 57`



「変数 `a` に `57` を代入する」

種類: 命令 (Command)

`a == 57`



「変数 `a` の値が `57` と等しいか？」

種類: 条件式 (Conditional Expression) - `True` または `False` (bool型) を返す

比較演算子 (Comparison Operators)

	演算子	意味
<code>a == b</code>	<code>a == b</code>	aとbが等しい
<code>a != b</code>	<code>a != b</code>	aとbが等しくない
<code>a > b</code>	<code>a > b</code>	aがbより大きい
<code>a >= b</code>	<code>a >= b</code>	aがb以上である
<code>a < b</code>	<code>a < b</code>	aがbより小さい
<code>a <= b</code>	<code>a <= b</code>	aがb以下である

 ***Bonus Tip*:** ``and``, ``or``, ``not`` といった論理演算子を組み合わせることで、``a >= 0 and a < 10`` のように複雑な条件も表現できます。

ロジックの構築②：退屈なタスクを繰り返す

`for`ループは、リストや範囲など、複数の要素を持つ「イテラブル」から要素を一つずつ取り出し、処理を繰り返します。

0から9までの整数を合計する

```
total = 0

# nを0から9まで順番に変化させながら、ブロック内の処理を繰り返す
for n in range(10):
    total += n # 「total = total + n」と同じ意味
print(total)
```

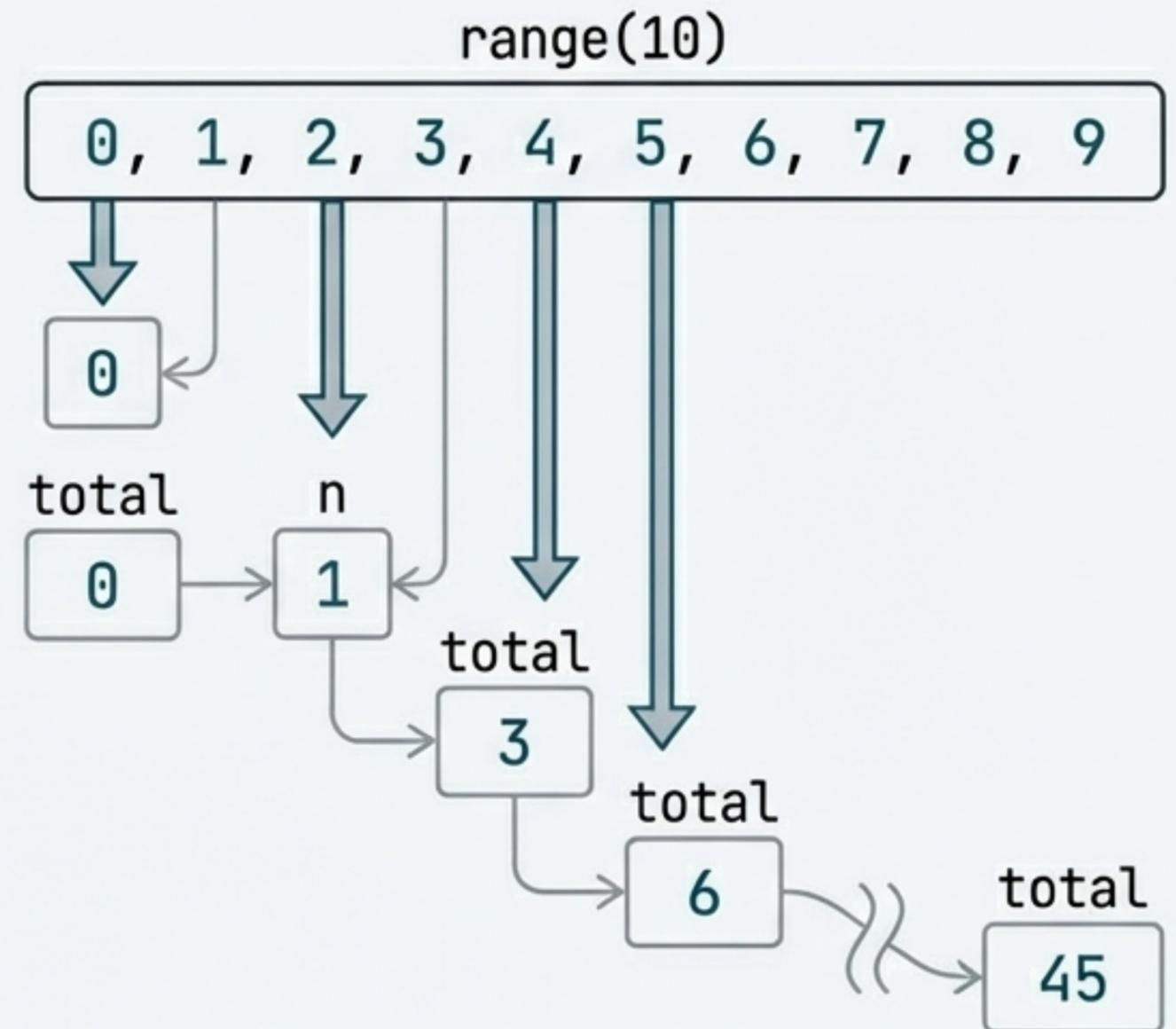
Execution Result

`45`

Syntax Breakdown

- `for n in range(10):`: `range(10)`は0から9までの整数を生成するイテラブル。`n`がその値を順番に受け取る。
- `total += n`: ループの各ステップで実行される処理。ここも**インデント**が必要です。

Pro-Tip: `for n in [2, 1, 3]:`のようにリストを使うと、`n`は`2`、`1`、`3`の順に変化します。



もう一つの繰り返し：条件が満たされるまで続ける `while`

`while`ループは、指定された条件が `True` である限り、処理を無限に繰り返します。

`for`ループと同じく、0から9までを合計する

```
total = 0
n = 0

while n < 10:
    total += n
    n += 1 # この行がないと無限ループに陥る！

print(total)
```



whileループでは、ループを終了させるための**条件変化**（例：`n += 1`）を**コード内に必ず記述**する必要があります。これを忘れると、プログラムが永遠に終了しない「無限ループ」を引き起こします。

自分だけの道具作り：処理をまとめる「関数」

頻繁に使う処理を一つの「関数」として定義しておくことで、コードを簡潔にし、再利用しやすくします。

```
# 入力xに対して 2 * (x + 1) を計算して返す関数fを定義
def f(x):
    y = x + 1
    return 2 * y

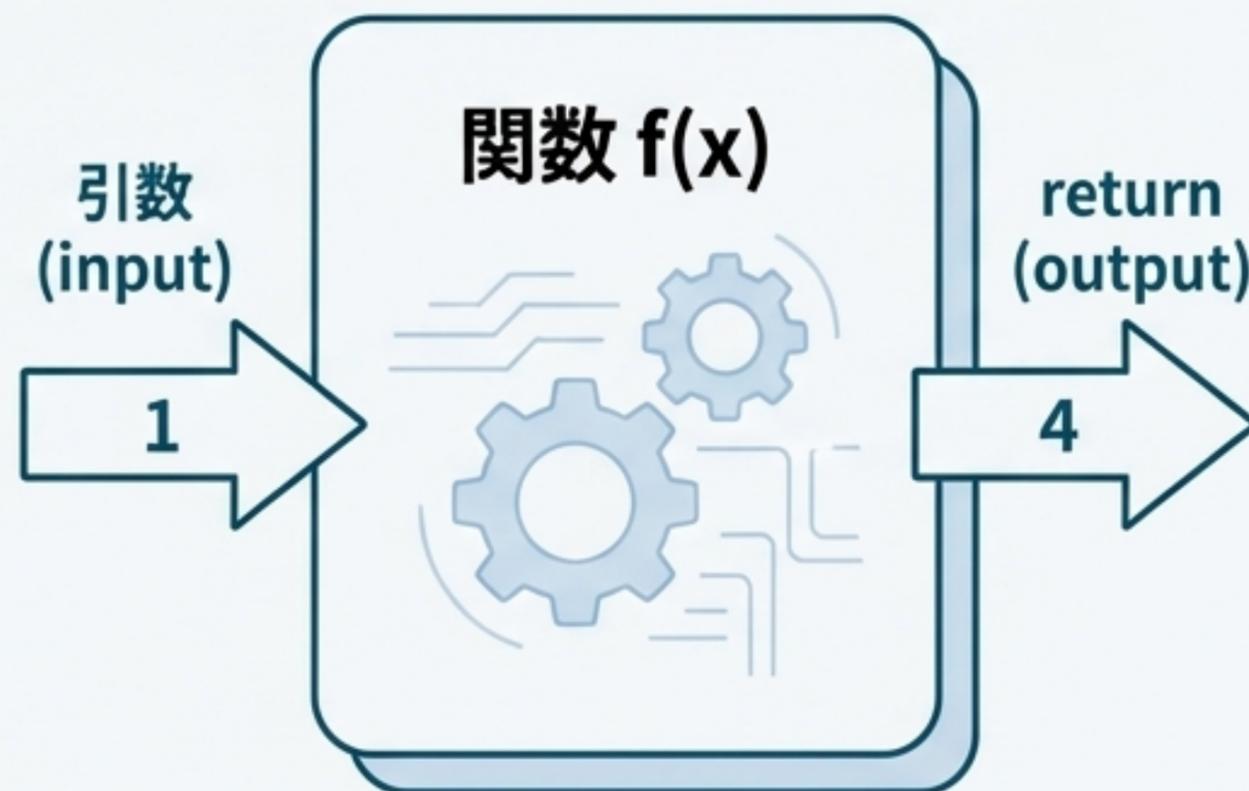
# 定義した関数を呼び出す
print(f(1))
```

Execution Result

4

Syntax Breakdown

- `def f(x):` f という名前の関数を定義 (define) します。
(x) は引数 (argument) と呼ばれる関数の入力です。
- ... 関数の処理内容。ここもインデントが必要です。
- `return 2 * y` 関数の出力 (output) となる値を返します。



飛躍的に効率を上げる：専門家の道具箱「ライブラリ」

プログラム開発者向けに、便利で高度な機能をまとめたものです。
Python の最大の強みは、この豊富なライブラリ・エコシステムにあります。



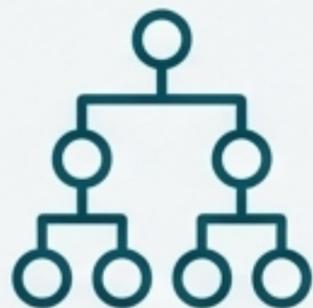
NumPy

配列操作、特に線形代数の演算を高速に実行。



pandas

データフレーム操作や欠損値処理など、表形式データの管理に必須。



scikit-learn

様々な機械学習アルゴリズムを実行。



Matplotlib, seaborn

データをグラフとして高品質に可視化。

How to Use Libraries

1. インストール (Install)

最初に一度だけ、コンソールでコマンドを実行します。

```
>> pip install numpy
```

2. インポート (Import)

プログラムの冒頭で、使用するライブラリを読み込みます。

```
import numpy as np
```

実践例：NumPyで配列データを一括処理する

配列 `[1.0, 5.0, 3.0]` の合計、平均値、中央値を計算します。



```
# NumPyライブラリを「np」という短い名前でインポート
import numpy as np

# NumPyの配列を作成
a = np.array([1.0, 5.0, 3.0])

# NumPyの便利な関数を使って計算・表示
print(np.sum(a))      # 配列の要素の和
print(np.mean(a))    # 配列の要素の平均
print(np.median(a))  # 配列の要素の中央値
```



Execution Result

```
9.0
3.0
3.0
```



Key Takeaway

NumPyのようなライブラリを使えば、本来ならループ処理が必要な計算も、簡潔な一行のコードで実行できます。

未来のワークフロー：AIアシスタントとしてのLLM活用

ChatGPTのような大規模言語モデル(LLM)は、自然言語の指示から非常に正確なPythonコードを生成できます。特に、使い慣れないライブラリの機能を調べたい場合に強力なパートナーとなります。



あなた (You): Matplotlibで関数 $y = x^2$ のグラフを描くソースコードを作ってください。

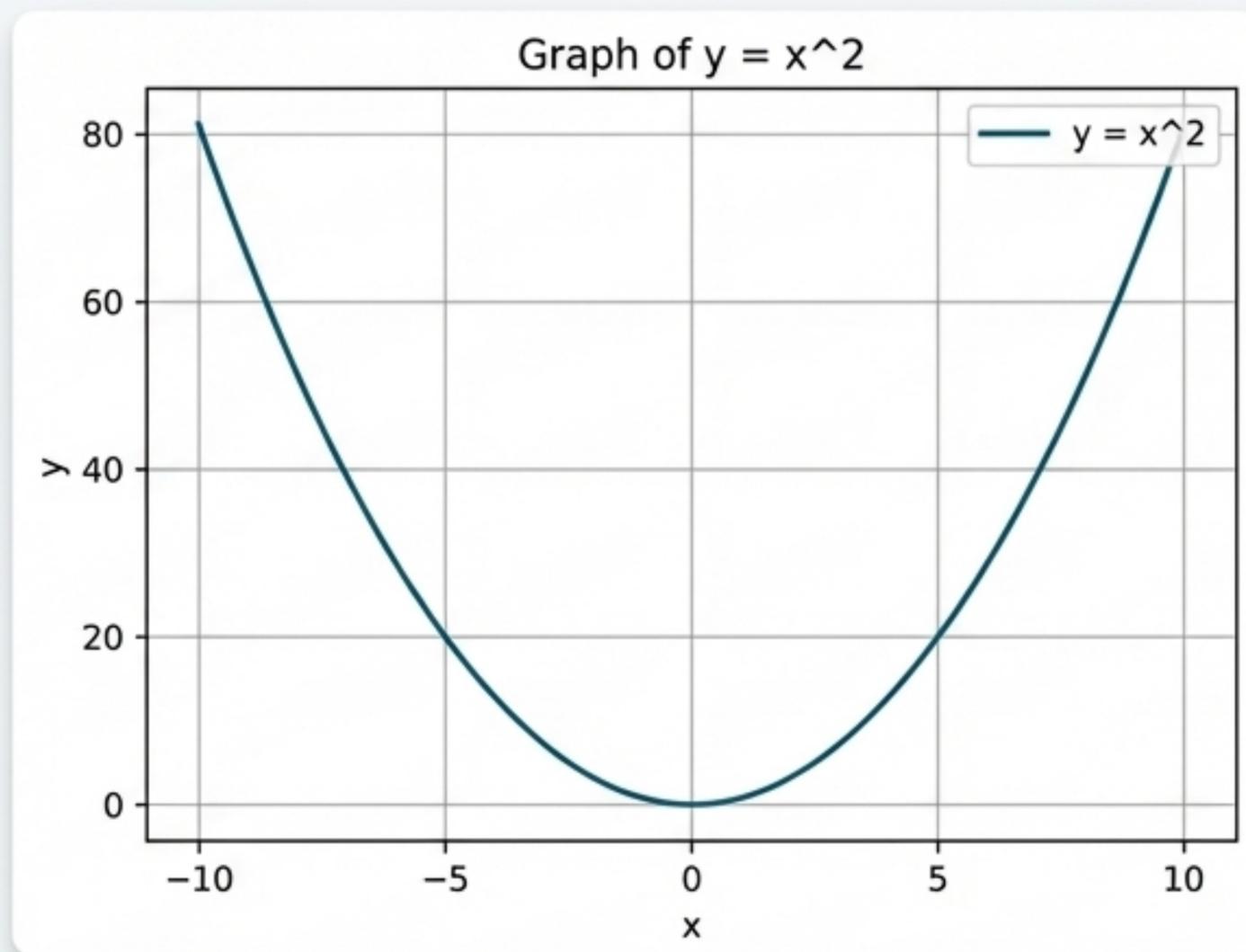


ChatGPT: はい、Matplotlibを使って関数 $y = x^2$ のグラフを描く基本的なソースコードを示します。

```
import numpy as np
import matplotlib.pyplot as plt

# x の範囲を設定 (-10 から 10 まで 200 点)
x = np.linspace(-10, 10, 200)
y = x ** 2 # 関数  $y = x^2$ 

# グラフの描画
plt.figure(figsize=(6, 4))
plt.plot(x, y, label="y = x^2")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Graph of y = x^2")
plt.grid(True)
plt.legend()
plt.show()
```



LLMを賢く使うための3つのルール

LLMは強力なアシスタントですが、最終的な責任は常にあなたにあります。



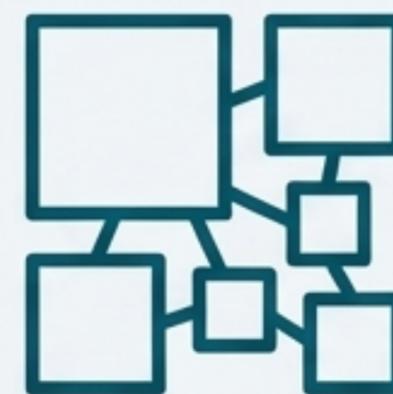
指示は具体的に (Be Specific)

漠然とした指示は誤解を招きます。
「どのライブラリを使い、どんな計算をしたいのか」を明確に伝えましょう。



必ず動作確認を (Always Verify the Code)

LLMが常に完璧なコードを生成するとは限りません。生成されたコードは必ず自分で実行し、期待通りに動作にするかを確認してください。



複雑な問題は分割する (Break Down Complex Problems)

長く複雑なコードを一度に作らせると、エラーの可能性が高まります。大きなタスクは複数の小さなステップに分け、それぞれについてコードを生成させましょう。

これがあなたの「データサイエンティストの工具箱」です

素材 (Materials)



変数と型 (Variables & Types)

基本ツール (Hand Tools)



演算子、条件分岐 (if)、繰り返し (for, while)、関数 (def)

パワーツール (Power Tools)



ライブラリ (Libraries)

AIアシスタント (AI Assistant)



大規模言語モデル (LLMs)

これらのツールを組み合わせることで、データサイエンスにおけるあらゆる課題に、効率的かつ創造的に取り組むことができます。

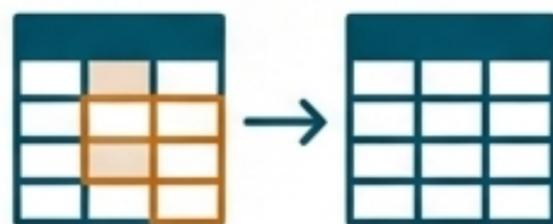
さあ、工具箱を使ってみよう

ここで学んだ知識とツールを使い、実際の問題に取り組んでみましょう。
LLMに質問しながら進めることを推奨します。



リストデータの分析 (List Data Analysis)

Pythonを用いて、リストに格納された数値データの平均値と中央値を計算するプログラムを作成してください。



pandasによる欠損値処理 (Handling Missing Values with pandas)

pandasライブラリを使い、CSVファイルからデータを読み込み、特定の列の欠損値を平均値で埋めるコードを書いてください。



Matplotlib/seabornによる可視化 (Visualization with Matplotlib/seaborn)

matplotlibやseabornを用い、データの散布図とヒストグラムを作成し、データの分布や関係性を視覚的に表現してください。