関数をつくってみる

- 複雑なプログラムを作る場合、
「ヤヤコシイ」部分を関数として切り分けると、
意外と簡単になることも

練習

- まずは、関数の実行イメージを身につける
- で、関数つかって面倒そうな問題を解いてみる
- 少しヤヤコシイ「引数の渡し方」の解説

簡単な関数の例

問題

- 何行かの文字列が与えられ、‘a’ を含む数が一番多い行を出力せよ
（数が同じ場合は、先に現れた行を出力すること）

フォーマット

```
N
Line1
…
lineN
```

入力例

```
3
abcdabcaba
aaabbaac
aaaccaac
```

正解

```
aaabbaac
```

簡単な関数の例

問題

- 何行かの文字列が与えられ、‘a’ を含む数が一番多い行を出力せよ
（数が同じ場合は、先に現れた行を出力すること）

フォーマット

```
N
Line1
…
lineN
```

入力例

```
3
abcdabcaba
aaabbaac
aaaccaac
```

正解

```
aaabbaac
```

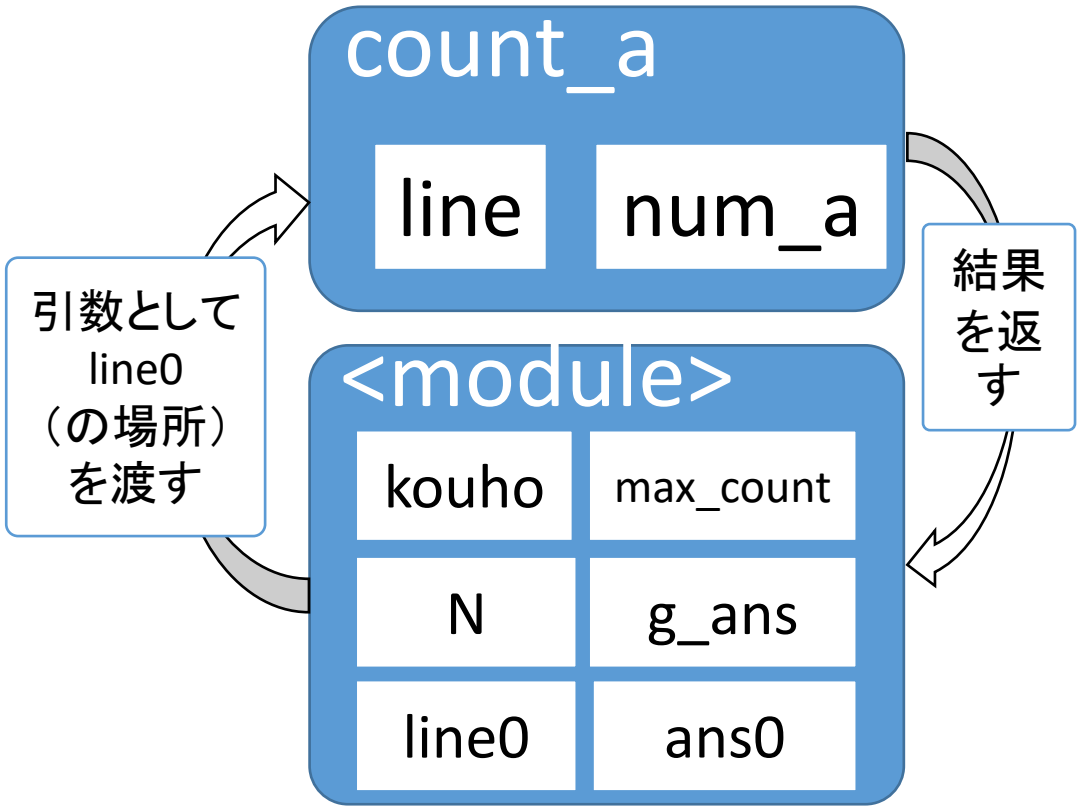
アプローチ

- 一行に含まれる ‘a’ の数をカウントする関数があれば簡単！

```
N = int(input())
kouho = ""
max_count = -1
for _ in range(N):
    line0 = input()
    ans0 = count_a(line0)
    if ans0 > max_count:
        kouho = line
        max_count = ans0
print(kouho)
```

関数呼び出し

- ‘a’ の数を数える関数 count_a(line) を作成



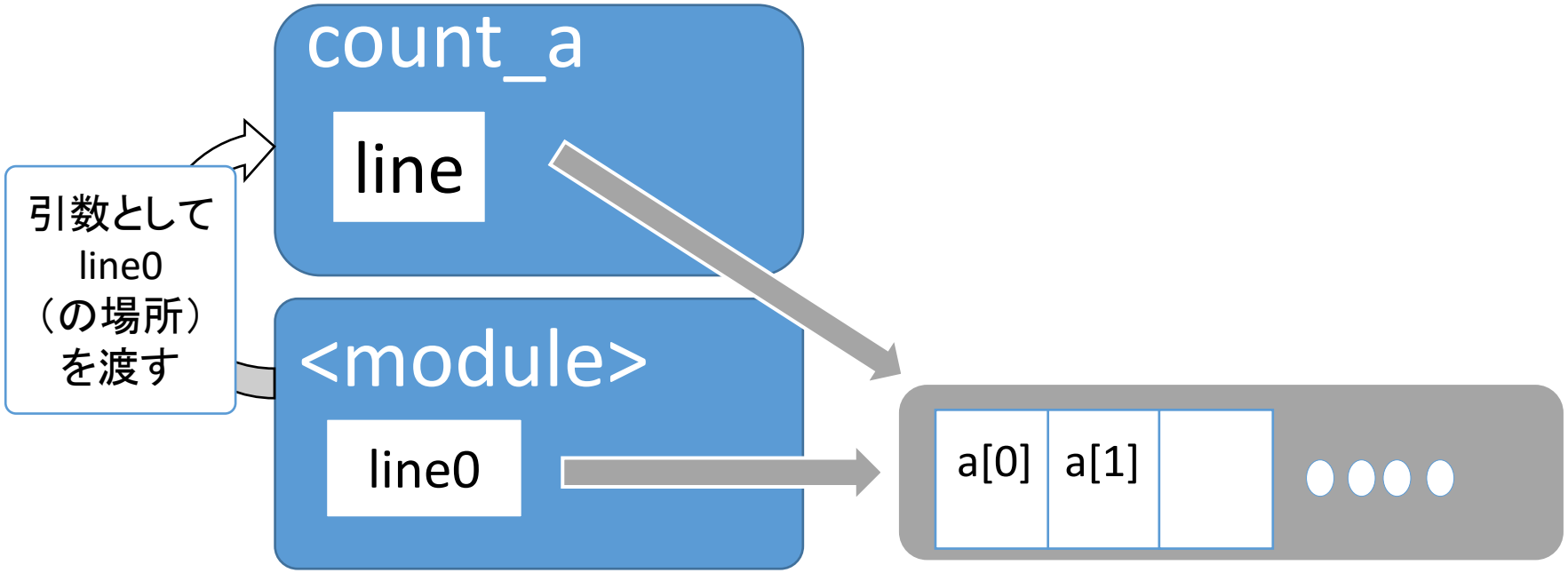

```
N = int(input())  # このあたりはグローバル変数
kouho = ""
max_count = -1
g_ans = 0

def count_a(line):  #line はローカル変数
    num_a = 0   # num_a, c はローカル変数
    for c in line:
        if c == 'a':
            num_a += 1
    g_ans = num_a  # 実はグローバル変数アクセス「じゃない」
    return num_a

for _ in range(N):
    line0 = input()    # このあたりもグローバル変数
    ans0 = count_a(line0)  # このあたりもグローバル変数
    print(line0, ans0, g_ans, max_count)
    if ans0 > max_count:
        kouho = line0
        max_count = ans0
print(kouho)
```

文字列やリストの扱い

- 今回、「1行の文字列」を引数で渡した
  - 別に文字列のコピーを作って渡したわけじゃない
  - 文字列やリストは、別のところにデータ構造があって、場所情報だけを受け渡したと考えてください

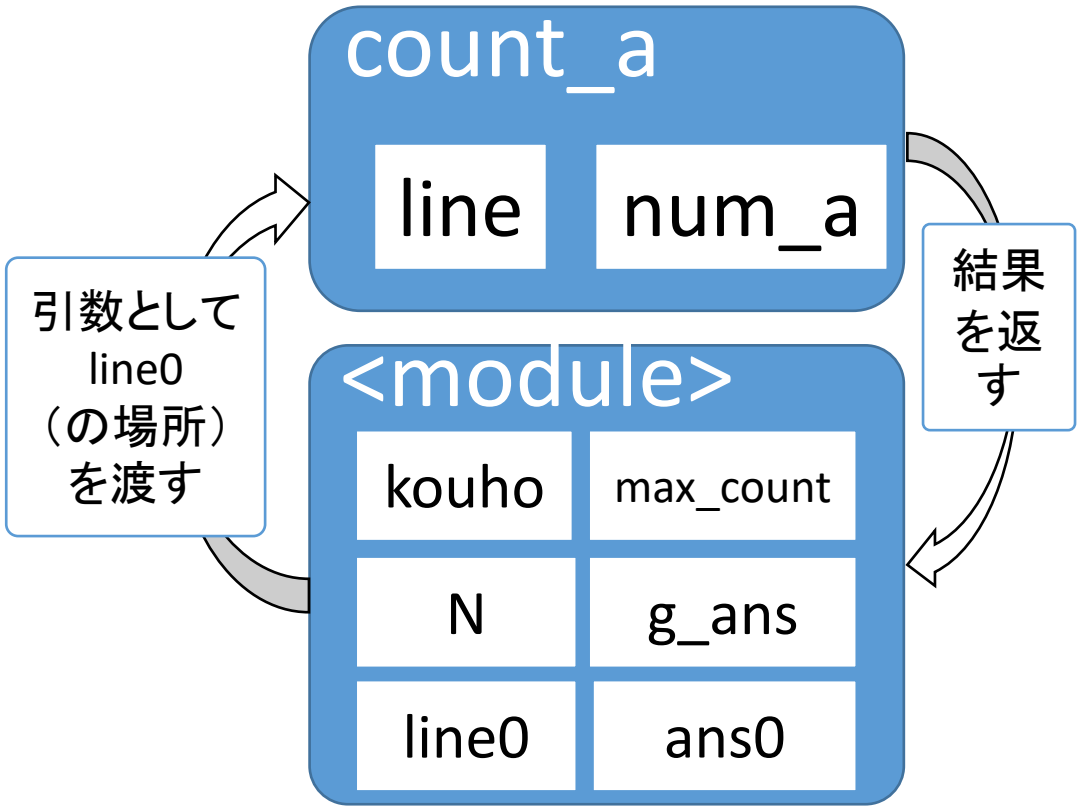


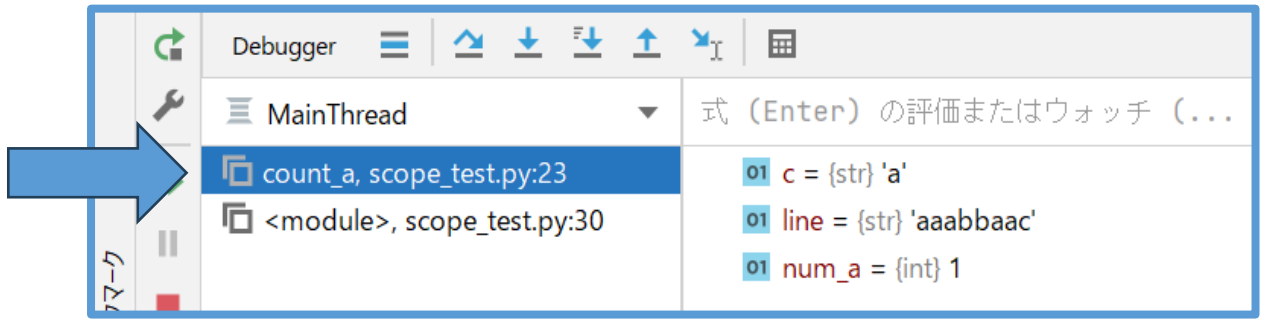
Python の変数

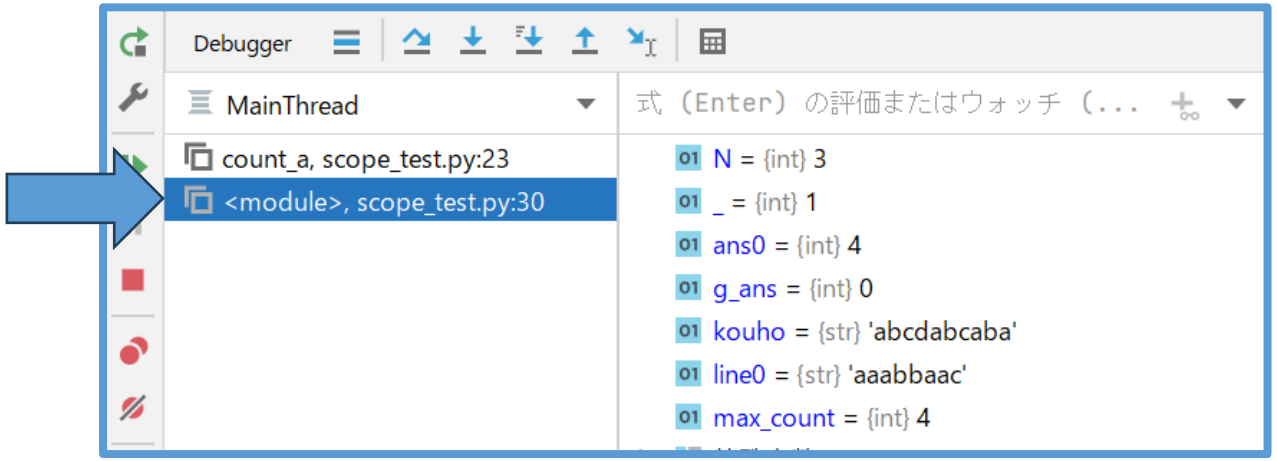
- ローカル変数: 関数内で代入された変数
  - 最初の代入で初期化、関数内でのみ読書き可能
  - 同名グローバル変数があっても、関数内での変数代入は、ローカル変数アクセスになる (global 文使わない限り)
- グローバル変数:関数外で宣言された変数
  - 関数外で読書き可能
  - 関数内でも読出し可能
  - 関数内でも global 文を使うと、書込み可能な変数として扱われる

デバッガ表示

- コールスタック：関数呼び出し関係表示
  - 各関数の局所変数やグローバル変数確認可能







ちょっとヤヤコシイ問題

第12回日本情報オリンピック 予選
C - 看板 (Signboard)

- 古い看板から文字を削って新しい看板を作る
  - 文字間隔は均等になるようにしたい
  - 古い看板は何枚か存在するので、大丈夫なものの数をカウント
- 入力例
  - 1行目：古い看板の枚数
  - 2行目：作りたい文字列
  - 3行目：古い看板の文字列
- 出力：使える古い看板の枚数

```
4
bar
abracadabra
bear
bar
baraxbara
```

baraxbara もあり

```
3
```

解けそうな気はする？

解きかけのプログラム

- とりあえず、入力を取り込むのは簡単

```
N = int(input())
target = input()
for _ in range(N):
    line = input()
    …
```

- さて、どうやって解く？
  - 意外に面倒そう。。。
- 単純な案
  - とりあえず、古い看板を1ずつチェック！
  - 文字の間隔を変えながら、使えるかどうかチェック！
  - 当然、開始位置も変えながらチェック！

ややこしいものは関数に切り分け

解きかけのプログラム

- とりあえず関数の作ることにしよう！
- def check()
  - target（作成したい文字列）
  - line（古い看板の文字列）

  の組が大丈夫かどうか判定

- 呼び出し側はスッキリ
  - あとは、関数つくるだけ

```
N = int(input())
target = input()
result = 0

for _ in range(N):
    line = input()
    if check():
        result += 1

print(result)
```

check 関数作るぞ

解きかけのプログラム

- とりあえず stride（文字間隔）を変えながら調べればいいよね
- といっても、開始位置変えたり、文字数分チェックしたりヤヤコシイ
- なので、ある stride を試す関数 try_stride(stride) に分業

```
def check():  # 古い看板（line）を 1 つチェック
    stride = 1
    while 1 + stride * (len(target) - 1) <= len(line):
        success = try_stride(stride)
        if success:
            return True
        stride += 1
    return False
```

try_stride(stride) を作るぞ

解きかけのプログラム

- これはこれで面倒
  - 開始位置を変える
  - target の文字数分確認しなくちゃ。。

さて、トライしてみよう！

参考
- 解きかけのプログラム
- 正解例 1, 2

このプログラムの実行時間は？

- この看板の問題、AtCoder では 10 sec, 256MB の制限
- 今回の解き方、ループがすごいけど大丈夫？？
  - 古い看板の数: N (≤ 100)
  - stride を変えながら: 場合によっては古看板の長さぐらい？ (≤ 100)
  - 開始位置も変えなくちゃ:場合によっては古看板の長さぐらい？ (≤ 100)
  - 新看板の長さ分のチェック: (≤ 25)
- ってことは、ループ内の命令列の実行回数は、
  最悪 25 x 100 x 100 x 100 回とかなるかも！！
  = 25 M 回！！

実行時間は、基本、
命令の実行回数や
メモリアクセス回数で決まる。

現実計算機のスペック

- CPU のクロック：数GHz
  - 1秒間に数G回の命令をこなす
    （1命令1nano sec以下）
  - 1000命令かかる処理でも、数M回こなす
  - つまり「1秒」で解くとは、
    数G回以内の命令で解くということ
- メモリ容量
  - 主メモリ数GBは当たり前
    - ➢でも、OS の領域も残してね
  - ハードディスクはTBオーダー
    - ➢でも、disk は遅いけどね

但し、主メモリランダムアクセスは、CPUより数十倍遅い

1K = 1000 = 10^3
1M =1000,000 = 10^6
1G = 1000,000,000 = 10^9
1T = 10^{12}

でも、キャッシュは数MB程度以下

量感覚イメージ

- 累乗: 2^{10}=1024≒K, 2^{20} ≒ M, 2^{30} ≒ G
- 階乗:
  - 1x2x3x4x5=120, 1x2x...x10=3.6Mぐらい
  - ..x13 で4G越える
- 32bit 符号付整数で表現できる値:　-2G ～ +2G
- 時間
  - 1秒に数G回の演算が可能
  - 1分:60秒、1時間:3.6K秒、1日:86.4K秒、
  - 1M秒で11日半ぐらい