# 電子計算機入門

池田, 大輔
九州大学附属図書館

# 電子計算機入門
# 第8回(06/23)

**池田 大輔**

z4id01in@cse.ec.kyushu-u.ac.jp

**附属図書館**

# 目次

- 前回課題の回答例
- エラーと例外
- デバッグ

# 前回課題の解答例

- 入力文字列中に現われる全ての部分文字列とその頻度を、頻度順に出力するプログラムを辞書を用いて作成せよ。

# 前回課題の解答例

- 入力文字列中に現われる全ての部分文字列とその頻度を、頻度順に出力するプログラムを辞書を用いて作成せよ。

```
import sys

str = sys.argv[1]

res = {} # res[substr]=n の形式で結果を格納する辞書

for b in range(len(str)): # 2 重 for ループで全ての部分文字列を生成

    for e in range(b+1, len(str)+1): # str[b:e] が新たな一台

        substr = str[b:e]

        if res.has_key(substr) :

            res[substr] += 1

        else :

            res[substr] = 1

array = [(res[key], key ) for key in res.keys()] # ソート用配列
```

# 前回課題の解答例(Cont.)

```
array.sort()

array.reverse()

for (v, n) in array: # 結果を出力

    print n, ":", v
```

# 配列 v.s. 辞書

- **配列と辞書の効率の違いの例**
- **第２回レポートを辞書と配列で実装し、**“large.txt” **と** $n = 3$ **で実行**

| | 1 | 2 |
|---|---|---|
| **辞書** | 41.85 | 36.57 |
| **配列** | 835.05 | 531.87 |

- **計算機環境：**

1. Power Book G4 (15"):PowerPC G4 1GHz/1G **バイト**
2. Power Book G4 (12"):PowerPC G4 1GHz/768M **バイト**

# なぜ配列は遅いのか？

- 「(*n*, *substr*) が配列の中にあるか否か？」が遅い

# なぜ配列は遅いのか？

- 「(*n*, *substr*) が配列の中にあるか否か？」が遅い

  → `if (n, substr) in array` は**不可**

  ∵ *n* の具体的な値は**不明**だから

# なぜ配列は遅いのか？

- 「(*n*, *substr*) が配列の中にあるか否か？」が遅い
  - → `if (n, substr) in array` は不可
  - ∵ *n* の具体的な値は不明だから
  - → *array* の要素を最初からチェックする必要あり

# なぜ配列は遅いのか？

- 「(*n*, *substr*) が配列の中にあるか否か？」が遅い

  → `if (n, substr) in array` は不可

  ∵ $n$ の具体的な値は不明だから

  → *array* の要素を最初からチェックする必要あり

- 配列の個数は $2\times$ 異なる「物」の数

  部分文字列の場合は最悪 $n(n-1)/2$　　$n^2$ 個存在する

  - $abcdef..z$ のようにすべて異なる文字の場合

# なぜ配列は遅いのか？

- 「(*n*, *substr*) が配列の中にあるか否か？」が遅い

  → `if (n, substr) in array` は不可

  ∵ $n$ の具体的な値は不明だから

  → *array* の要素を最初からチェックする必要あり

- 配列の個数は $2\times$ 異なる「物」の数

  部分文字列の場合は最悪 $n(n-1)/2$　　$n^2$ 個存在する

  - $abcdef..z$ のようにすべて異なる文字の場合

  → $n^2$ 個の各「物」に対し、配列全体のスキャン $(n^2)$ をするので総計 $n^4$ かかる

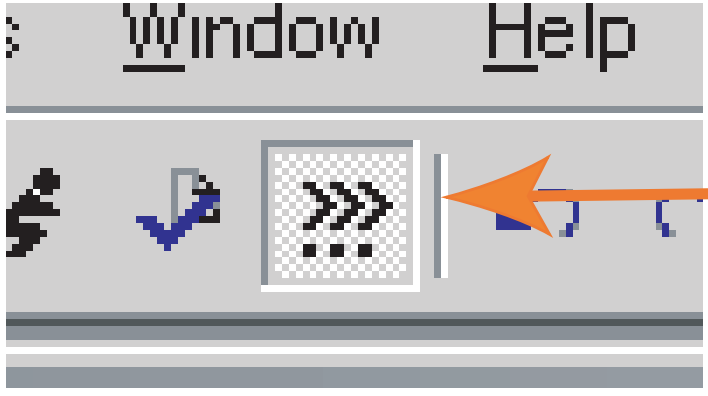
- 辞書の場合は全体で $n^2$ しかかからない

# $n^4$ と $n^2$ の効率

- 入力文字列の長さ (サイズ) が $n$ である
- 長さが $1000 = 10^3$ の文字列の場合
  - $n^4 = 10^{12} = 1$ 兆回
  - $n^2 = 10^6 =$ 百万回
  - 最近の CPU は数ギガヘルツ=20 億回／秒

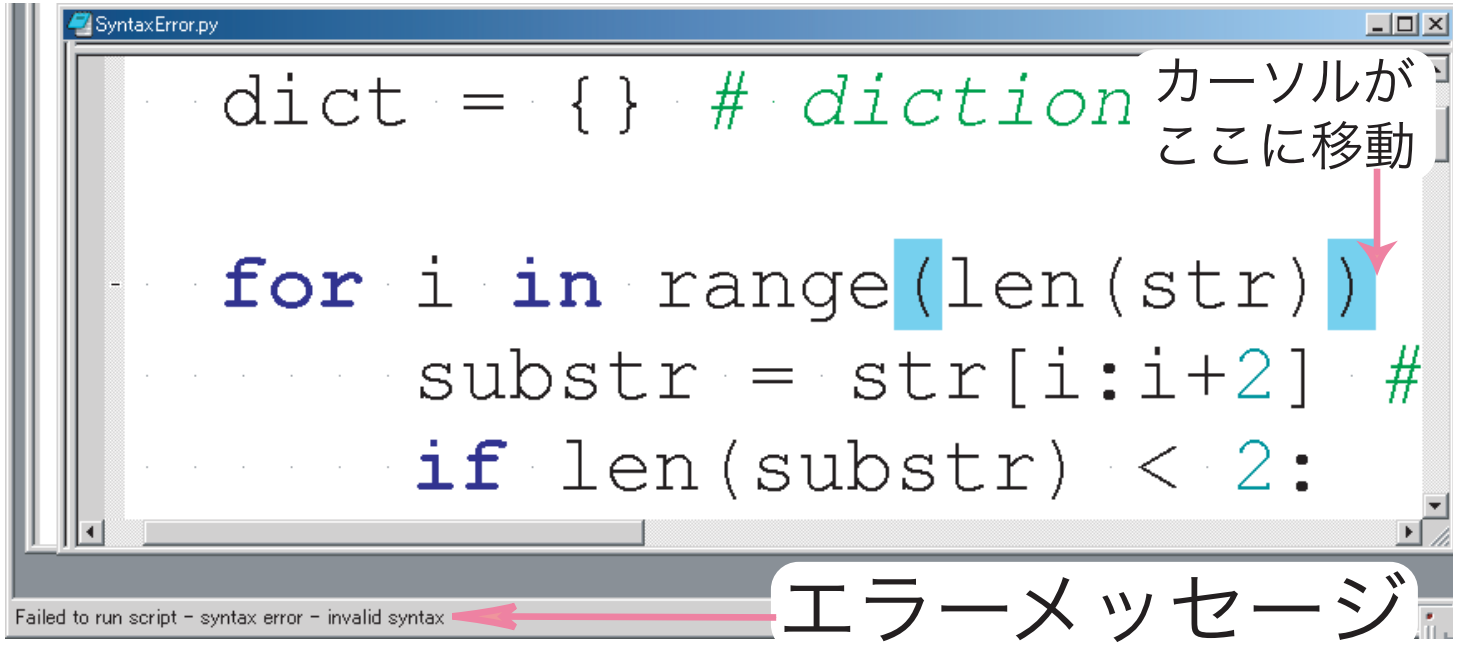
# Pythonでのエラー

- **構文的なエラー** (syntax error)
  - **“:” を忘れた、インデントが正しくない、など**
- **構文は正しいがエラーとなるもの**
  - **配列の範囲外にアクセスした、存在しない変数にアクセスした、など**
- **エラーメッセージは対話型ウィンドウに表示される**



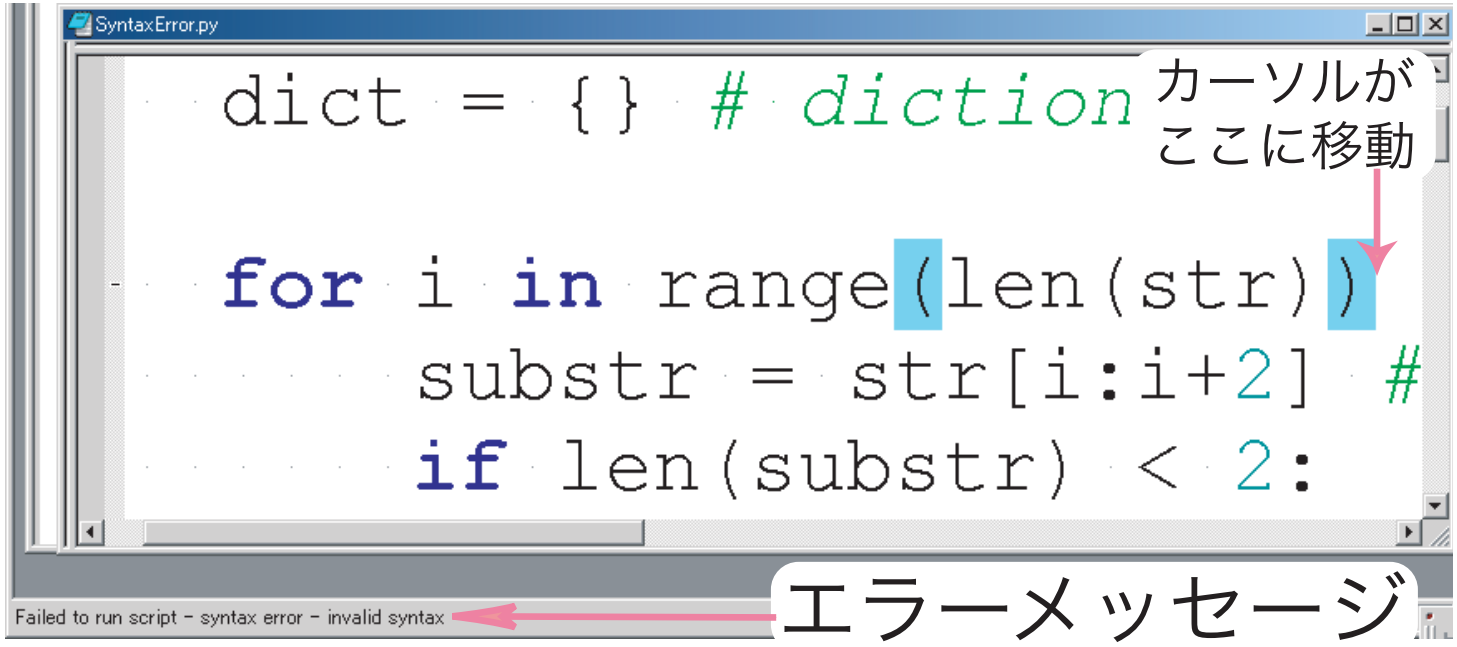
# 構文エラー

- エラーメッセージはウィンドウの一番下に表示



# 構文エラー

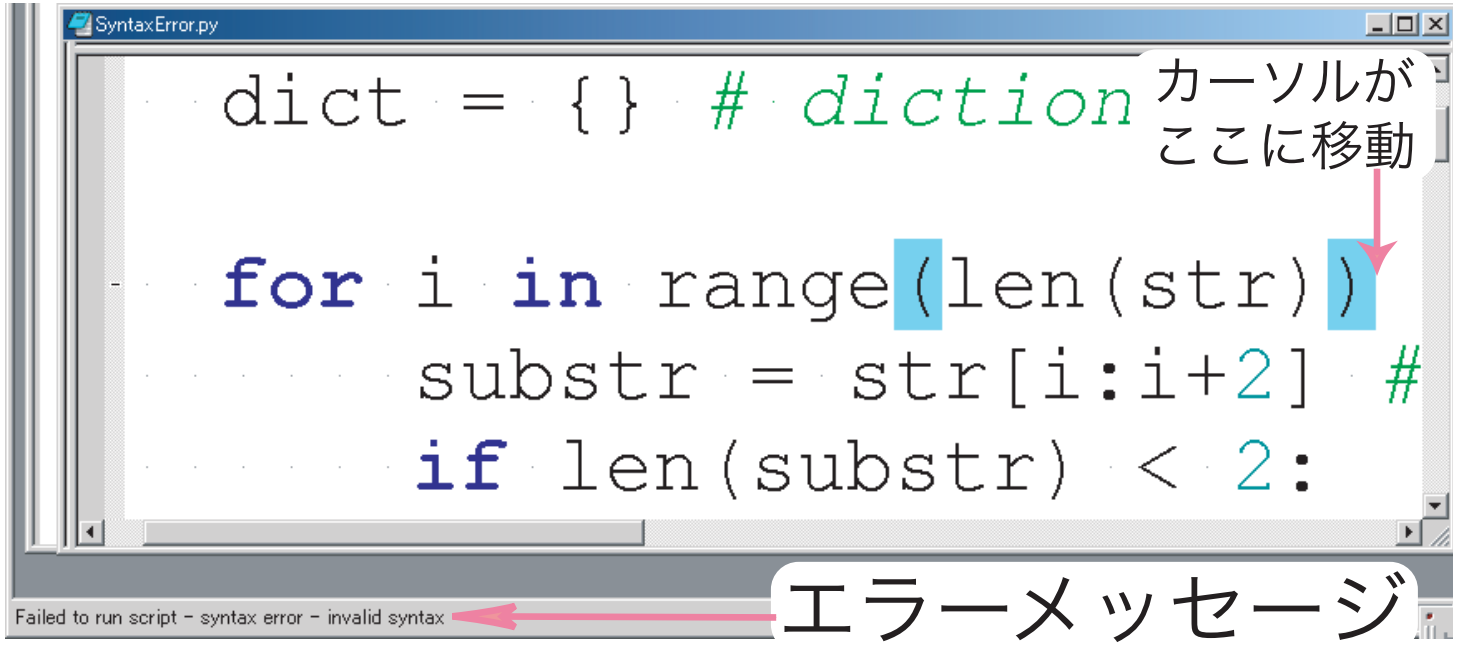
- エラーメッセージはウィンドウの一番下に表示



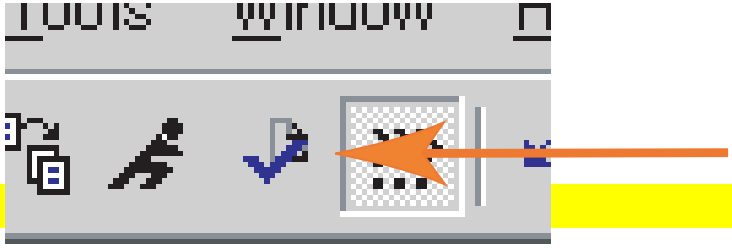

- 構文エラーが起きたときは実行はされないので、
エラーを解消してから再度実行する

# 構文エラー

- エラーメッセージはウィンドウの一番下に表示




- 構文エラーが起きたときは実行はされないので、エラーを解消してから再度実行する
- Pythonwin には構文チェックをする機能あり



# よくある構文エラー

- `if, while, for` などの最後の `:`(コロン) を忘れた
- インデントが正しくない
- `()` や `[]` の数が対応していない

# 実習：構文エラー

- “SyntaxError.py” をダウンロードして、実行しなさい
  - 前回課題の解答例と同じだが、構文的なエラーを意図的にいれてある。
  - 実行するには、一つの文字列を入力として与えること
- 正しく実行できるようにしなさい

今日の課題！

# よくある実行時エラー

- 構文的には正しいが実行するとエラーになるもの
- 存在しない変数にアクセスする
  - 授業中の質問は、ほとんどこの問題である

“=” の右辺に現われる変数は、それ以前に値を代入されている必要あり。つまり、それより前の=の右辺に現われていないといけない

---

```
res[substr] = res[substr] + 1
```

とする場合は、それより前に

```
res[substr] = ...
```

という文が必要

# デバッグ

- バグ：意図した動きと違う動きをする原因となる箇所
  - 例外や構文エラーも一種のバグ
  - ただし、プログラムは動くけど、意図した通りには動かないバグのほうが扱いにくい
- デバッグ⇔バグをなくすこと
  - 実行中に変数の値を出力して確認するのが基本的なデバッグの手法
  - 効率よくデバッグするためのソフト (デバッガ) もある

# print文によるデバッグ

- 動きがおかしい、何も出力されない

  → 途中で変数の値を print により出力させる

  - 手軽に利用できる
  - デバッグ後、print 部分をコメントアウトするか削除しないといけない
    - 余計な出力で本当の出力結果が見えにくくなるため

# assert文によるデバッグ

- 正しい主張を assert 文で書いておく
  - 手軽に利用できる
  - 主張にそって正しく動いているときは何も起こらず、違う動きのときに例外を起こす
    - コメントアウトする必要がない
    - プログラムを書いた時の意図が他人に伝わりやすい

# assert 文の使い方

- 述べたい主張を if 文の条件部分のようにかく

```
assert condition
```

# assert 文の使い方

- 述べたい主張を if 文の条件部分のようにかく

```
assert condition
```

- 例
  - 変数 var は必ず偶数である

```
assert var % 2 == 0
```

  - 引数はちょうど 2 個である

```
assert len(sys.argv) == 2
```

# 付録：関数

- プログラムのうち、特定の機能に名前をつけて再利用可能にしたもの
- 入力を受けとり、値を返す (出力する)

---

```
def function(x, y):
    '''
    ここの文字列は function の説明
    一種のコメント文字列
    '''
    ここに function の動作記述
    function の範囲はインデント
    return(var) # 配列を返してもよい
```

# 付録：関数の使用例

```
def substrings(s):
    '''文字列sのすべての部分文字列を生成する'''
    result = []
    for i in range(len(s)):
        for j in range(i,len(s)):
            if s[i:j+1] not in result:
                result.append(s[i:j+1])

    return(result) # 関数の定義終了
import sys # ここから``メイン''部分
str=sys.argv[1]
# 全ての部分文字列を関数により生成
substrings=substrings(str)
# 個々の部分文字列を処理
for substr in substrings:
    ...
```