

## 電子計算機入門

池田, 大輔  
九州大学情報基盤センター

<https://hdl.handle.net/2324/6097>

---

出版情報 : 2003  
バージョン :  
権利関係 :

# 電子計算機入門 第8回

池田 大輔

daisuke@cc.kyushu-u.ac.jp

情報基盤センター

# 目次

- 関数呼びだし (Cont.)
  - 前回課題の回答例
- 辞書
- 第2回レポート課題予告

# 復習：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
```

# 前回課題の解答例

```
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  
ss1=substrings(sys.argv[1])  
ss2=substrings(sys.argv[2])
```

# 前回課題の解答例 (Cont.)

# 共通部分文字列とその長さを格納

```
common = [(len(i), i) for i in ss1 if i in ss2]
```

```
common.sort() # 長さでソート
```

```
common.reverse() # 長さでソート
```

# 最初の要素 (len(i), i) の最初の要素が最大の長さ

```
MAX = common[0][0]
```

```
res = [i for (l, i) in common if l == MAX]
```

```
print res
```

# 関数の使用例

## ■ 使用方法を出力するだけの関数

```
def Usage():
```

```
    print "Usage:  ...."
```

# 値を返す必要がないので return がない

# 関数の使用例

## ■ 使用方法を出力するだけの関数

```
def Usage():  
    print "Usage: ...."
```

# 値を返す必要がないので return がない

## ■ 途中で値を返す関数

```
def func(x):  
    if ....: return(x)  
    else ....: return(x)
```



# 関数の呼び出し方

- file.py で `def function` した場合

# 関数の呼びだし方

- file.py で `def function` した場合
- `def` したのと同じファイルから  
`function(var1, var2)`

# 関数の呼びだし方

- file.py で `def function` した場合

- `def` したのと同じファイルから

```
function(var1, var2)
```

- 違うファイルから

```
import file
```

```
file.function(var1, var2)
```

# 関数の呼びだし方

- file.py で `def function` した場合

- `def` したのと同じファイルから

```
function(var1, var2)
```

- 違うファイルから

```
import file
```

```
file.function(var1, var2)
```

- **プログラムの再利用が容易**

# 関数の呼びだし方

- file.py で `def function` した場合

- `def` したのと同じファイルから

```
function(var1, var2)
```

- 違うファイルから

```
import file
```

```
file.function(var1, var2)
```

- **プログラムの再利用が容易**

- 例

```
import algorithm1
```

```
ss1 = algorithm1.substrings(s1)
```

```
ss2 = algorithm1.substrings(s2)
```

# import の詳細

- 以下のようなファイル algorithm1.py の場合

---

```
def substrings():  
    ...  
# ここまでが関数の定義  
# ここからがメイン  
import sys  
s1=sys.argv[1]  
s2=sys.argv[2]
```

---

import algorithm1 をするとメインも実行される!!

- substrings を再利用したいただけなのに...
- “IndexError: list index out of range” とエラーがでる

# ■ 実習：別ファイルの関数呼び出し

- 新たにスクリプト “funcall.py” を作成し ([File] → [New])、一つの文字列を引数として受けとり、この部分文字列をすべてを出力するプログラムを作りなさい
  - 先週作った “algorithm1.py” の `substrings` という関数を利用すること

# ■ 実習：別ファイルの関数呼び出し

- 新たにスクリプト “funcall.py” を作成し ([File] → [New])、一つの文字列を引数として受けとり、この部分文字列をすべてを出力するプログラムを作りなさい
  - 先週作った “algorithm1.py” の `substrings` という関数を利用すること

---

```
import algorithm1
import sys
print algorithm1.substrings(sys.argv[1])
```

---



# import の詳細 (2)

- “algorithm1.py” を以下のように変更する

```
def substrings():  
    # 関数の定義  
  
if __name__ == '__main__':  
    # メイン部分はこの if 文中におさめる  
    import sys  
    s1=sys.argv[1]  
    s2=sys.argv[2]
```

- if 文の中はこのファイルを実行 ([Run]) したときに実行されるが、import algorithms では**実行されない**

# ■ 実習：別ファイルの関数呼び出し

- “algorithm1.py” のメイン部分を変更し、“funcall.py” から `substrings` を利用可能にした上で、“funcall.py” を実行せよ

---

```
def substrings(s):  
    # 中身は同じ  
  
if __name__ == '__main__':  
    import sys  
  
    s1=sys.argv[1]  
    s2=sys.argv[2]  
  
    ss1=substrings(s1)  
    ss2=substrings(s2)  
  
    common = [] # 共通部分文字列とその長さを格納
```

# 頻度カウント

- 「何」が「何回」現われたか
- 単なる配列では「何回」しか記録できない  
多次元配列などが必要

count = [ (n<sub>1</sub>, '物 1'), (n<sub>2</sub>, '物 2'), ... ]

- '物' として部分文字列、n<sub>i</sub> としてその頻度など
- n<sub>i</sub> で順番をつけたいので、これを先にする
  - ▼ 先週の max, sort などを参照

# 頻度カウンタの基本的な方法

- 結果を記録する配列 `count = []` を用意
- 新たな部分文字列 `str` が来た
- **if** `str` はすでに `count` に記録済み  
 $(n, str)$  を  $(n + 1, str)$  へ
- **else**  
`count.append(1, str)`

# 頻度カウンタの基本的な方法

- 結果を記録する配列 `count = []` を用意
- 新たな部分文字列 `str` が来た
- **if** `str` はすでに `count` に記録済み ← ここに要する時間が?  
 $(n, str)$  を  $(n + 1, str)$  へ
- **else**  
`count.append(1, str)`

# 頻度カウンツの基本的な方法 (2)

# 'str' の頻度を調べる場合

```
for i in range(len(count)):
```

```
    (n, v) = count[i]
```

```
    if == str: # すでに記録済みの場合
```

```
        count[i] = (n+1, v) #値を増やす
```

```
        break)
```

```
else: # 記録していない場合
```

```
    count.append((1, str))
```

---

## 頻度カウンタの基本的な方法 (2)

# 'str' の頻度を調べる場合

```
for i in range(len(count)): 全部にアクセス
```

```
    (n, v) = count[i]
```

```
    if == str: # すでに記録済みの場合
```

```
        count[i] = (n+1, v) # 値を増やす
```

```
        break)
```

```
else: # 記録していない場合
```

```
    count.append((1, str))
```

---

## ■ 頻度カウンタの基本的な方法 (2)

# 'str' の頻度を調べる場合

```
for i in range(len(count)): 全部にアクセス
    (n, v) = count[i]
    if v == str: # すでに記録済みの場合
        count[i] = (n+1, v) # 値を増やす
        break)
else: # 記録していない場合
    count.append((1, str))
```

---

■ 上の for の代わりに if v in count: とは **できない**

- count の要素は v ではなく (n, v) だが、n の値は具体的には予測できない!



# 実習：頻度カウント

- 長さ 2 の部分文字列の頻度をカウントするプログラム  
“CountByList.py” をコピーして、頻度カウントを行なう  
部分を完成させなさい

[share] → [teachers] → [z3id01in]

- “# check if 'substr' is in count” 部分を完成させる
- 適当な引数を与えて、実行させなさい

# 実習：頻度カウント

- 長さ 2 の部分文字列の頻度をカウントするプログラム “CountByList.py” をコピーして、頻度カウントを行なう部分を完成させなさい

[share] → [teachers] → [z3id01in]

- “# check if 'substr' is in count” 部分を完成させる
- 適当な引数を与えて、実行させなさい

```
for i in range(len(count)):  
    (n, s) = count[i]  
    if s == substr:  
        count[i] = (n+1, s)  
        break  
else: # 'break' が呼ばれなかった時  
    count.append((1, substr))
```

# すでに記録済みか？

- count が多次元配列の場合
- 配列のサイズは  $2 \times$  異なる「物」の数  
部分文字列の場合は最悪約  $n(n-1)/2 = n^2$  個存在する
  - *abcdef..z* のようにすべて異なる文字の場合
- 配列中に特定の値があるかどうかは、配列の最初からスキャンするしかない  
→  $n^2$  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

- マニュアル「2.1.6 マッピングタイプ」やチュートリアル「5.4 辞書」を参照
- '物' と値の対応表
  - ハッシュ、連想配列などとも呼ばれる
  - 「記録済みかどうか」が一瞬で分かる
- キーで添字づけした複数のデータ
- キーは文字列や数字など、変更不能であれば何でもよい
  - 配列は変更可能なので、キーにはできない
  - 配列のキーは、0以上の連続な自然数

# ■ 頻度カウンタの基本的な方法：辞書

```
dict = {} # 空辞書を用意
# 新たな'物'v がきた場合
if dict.has_key(v): # v がキーとして存在するか?
    dict[v] += 1 # v の値を増やす
    # 個々の値へは dict['キー'] でアクセス
else:
    dict[v] = 1 # 始めての時は初期値(1)
```

---

# ■ 頻度カウンタの基本的な方法：辞書

```
dict = {} # 空辞書を用意
# 新たな'物'v がきた場合
if dict.has_key(v): # v がキーとして存在するか?
    dict[v] += 1 # v の値を増やす
    # 個々の値へは dict['キー'] でアクセス
else:
    dict[v] = 1 # 始めての時は初期値 (1)
```

---

- ループが不要なので効率が非常によい

# 実習：頻度カウントと辞書

- “CountByDict.py” をコピーして残りを完成させなさい
  - “# check if 'substr' is in count” 部分を完成させる

# 実習：頻度カウントと辞書

- “CountByDict.py” をコピーして残りを完成させなさい
  - “# check if 'substr' is in count” 部分を完成させる

```
if dict.has_key(substr):  
    dict[substr] = dict[substr] + 1  
else:  
    dict[substr] = 1
```



# 配列 v.s. 辞書

- 配列が便利な場合
  - 自然な番号づけがある
  - この番号 (のみ) でデータにアクセスする
  - 番号が連続している (連続してアクセスする)
- 例えば、すべての学生に関するデータ
  - 学籍番号があり、番号の抜けもない
- 逆に、
  - どのデータへアクセスするかわからない
  - 頻繁にいろんなデータへアクセスする場合は辞書がよい

# 全辞書データへアクセス

- 辞書は**順番がない**ので for 文で**処理できない**

→すべての“キー”、“値”、“(キー、値)のペア”の配列を返す関数を利用

```
for key in dict.keys(): # キーでアクセス
    print key, dict[key]
```

---

```
for val in dict.values(): # 値でアクセス
    print val # この場合、キーは不明
```

---

```
for (key, val) in dict.items(): # (キー、値)
    print key, val
```

# 実習：辞書データへアクセス

- “CountByDict.py” の “# print all substrings” 部分を完成させなさい

# 実習：辞書データへアクセス

- “CountByDict.py” の “# print all substrings” 部分を完成させなさい

---

```
for (key, val) in dict.items():  
    print key, " occurs ", val, " times"
```

# 今日の課題

- 頻度カウントするプログラム“CountByDict.py”を改良し、入力として整数  $n$  も受けとり、長さ  $n$  の部分文字列の頻度をカウントするようにしなさい
  - 現状の“CountByDict”では、長さは2に固定している

# 第2回レポート予告

- 以下を行なうプログラムを作成しなさい
  - ファイルのデータを読み込んで長さ  $n$  の部分文字列の頻度をカウント
  - 頻度によるソートして頻度の高い順に出力