電子計算機入門

池田, 大輔
九州大学情報基盤センター

電子計算機入門
第10回

池田 大輔

daisuke@cc.kyushu-u.ac.jp

情報基盤センター

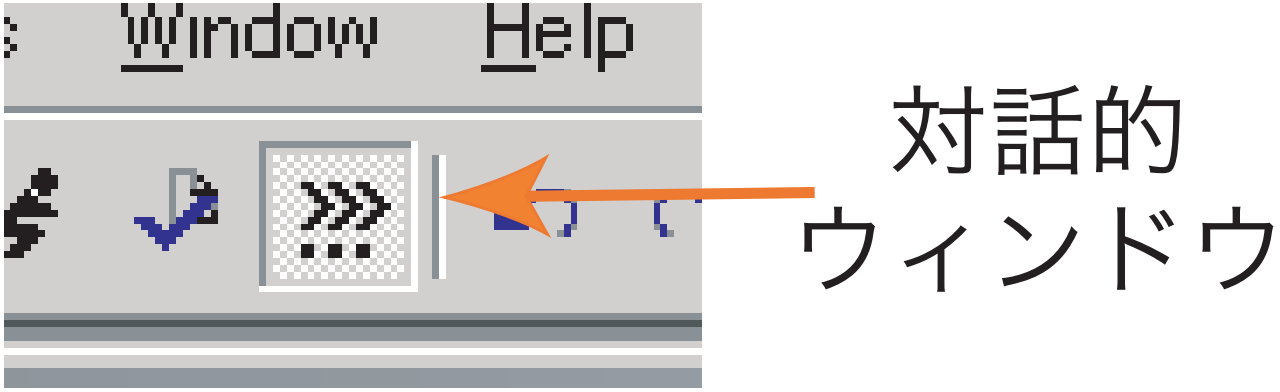
目次

- Python でのエラー
  - エラーの具体例
  - 例外処理
- デバッグ
- 前回課題の回答例

Pythonでのエラー

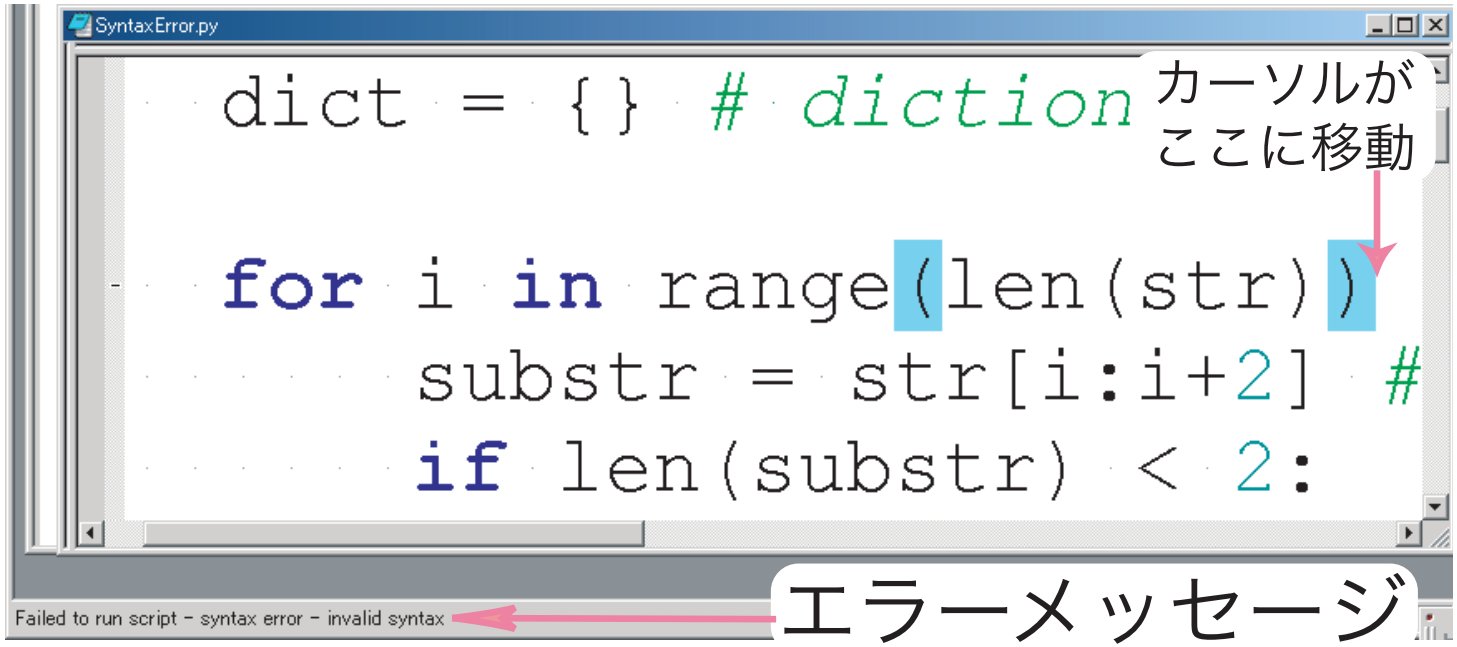
- Pythonでは例外 (exception) と呼ばれる

  リファレンスマニュアル「2.2 組み込み例外」
- 構文的な例外
  - ":" 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
  - >>> のあるウィンドウのこと

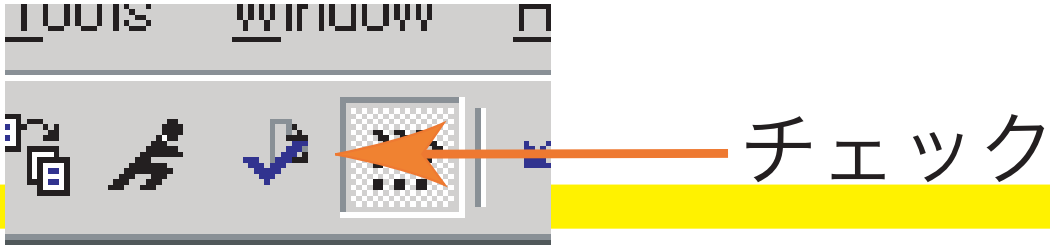


構文エラー

- 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



実習：例外

- “SyntaxError.py” をダウンロードして、実行しなさい
  - このプログラムは、長さ２の部分文字列を頻度の高い順に出力することを意図している
  - 引数に文字列を１つ与えて実行しなさい
  - 構文的なエラーを意図的にいれてある
- 正しく実行できるようにしなさい

例外の具体例

- SyntaxError
  - エラーの場所 (“^”) と原因を表示

```
  File "<string>", line 5
    for i in range(len(str))
                           ^
SyntaxError: invalid syntax
```

- これは for の後に “:”(コロン) を忘れている

例外の具体例(2)

- IndentationError

```
  File "<string>", line 8
    continue
             ^
IndentationError: expected an indented block
```

→ if 文の後なのにインデントしていない

```
  File "<string>", line 28
    list.sort()
               ^
IndentationError: unindent does not match any
```

→上下の文とインデントがずれている

実習：例外

- “Exception.py” をダウンロードし、実行しなさい
- 構文的ではないエラーを意図的にいれているので、これらのエラーを解消しなさい

例外の具体例

- TypeError

```
Traceback (most recent call last):
  File "Exception.py", line 16, in ?
    hash = countsubstrings(str)
  File "Exception.py", line 7, in countsubstr
    dict[substr] += 1
TypeError: unsubscriptable object
```

→ dict={}という初期化を忘れたから

例外の具体例(2)

- KeyError

```
Traceback (most recent call last):
  File "Exception.py", line 17, in ?
    hash = countsubstrings(str)
  File "Exception.py", line 8, in countsubstr
    dict[substr] += 1
KeyError: aa
```

注：“aa”は入力文字列の一部

→“aa”をキーとする`dict['aa']`という値がないのに

```
    dict[substr] = dict[substr] + 1
```

としてアクセスしてしまった

例外の処理

- 何もしなければ例外によりプログラムは終了させられる
- 適当な処理をして、そのまま次に進みたいこともある

  起こり得る例外のうち、処理したい例外を `try`, `except` 文で処理する
- 書式

```
try:
    # 例外が起こりそうな文
    # try の範囲はインデント
except 例外名:
    # 例外が起こった時の処理
    # 上の try 中で例外が起こらなければ、
    # ここは無視される
```

例外の処理(Cont.)

- 例：引数で与えた文字列を整数に変換する

```
n = int(sys.argv[1])
```
←数字でないと例外発生

例外の処理(Cont.)

- 例：引数で与えた文字列を整数に変換する

```
n = int(sys.argv[1]) ←数字でないと例外発生
```

```
try: # コロンに注意
    n = int(sys.argv[1])
except ValueError: # コロンに注意
    print 'Argument must be integer'
    sys.exit() # 強制終了
```

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
```

type 関数

- 変数などのタイプ (型) を返す関数

type関数

- 変数などのタイプ(型)を返す関数

```
assert(type(sys.argv[1]) == type(""))
# 最初の引数は文字列である、という主張
# ""は空の文字列
```

type関数

- 変数などのタイプ(型)を返す関数

```
assert(type(sys.argv[1]) == type(""))
# 最初の引数は文字列である、という主張
# ""は空の文字列
assert(type(sys.argv[1]) == type(0))
# 最初の引数は整数である、という主張
```

type関数

- 変数などのタイプ (型) を返す関数

```
assert(type(sys.argv[1]) == type(""))
# 最初の引数は文字列である、という主張
# ""は空の文字列
assert(type(sys.argv[1]) == type(0))
# 最初の引数は整数である、という主張
assert(type(sys.argv[1]) == type(0.0))
# 最初の引数は実数である、という主張
```

実習：assert文の使い方

- “assert.py” をダウンロードしなさい
  - このプログラムは最長共通部分文字列を解くつもりで書いたものである
  - ただし、うまく動いていないので、`print` 文や `assert` 文でデバッグし正しく動作させなさい
    →これが今日の課題

第2回レポートの注意

- 以下を行なうプログラムを作成しなさい
  - 入力：文字列を 1 つと整数 n
  - 出力：この文字列の長さ n の部分文字列
    ただし、頻度の高い順に出力する
- 関数を使う場合は、その関数は他のファイルからも利用可能でなければならない

前回課題の解答例

- 長さ 3 の部分文字列の頻度をカウントし、頻度順に出力

```
def countsubstrings(str):
    hash = {}
    for i in range(len(str)):
        substr = str[i:i+3]
        if len(substr) < 3:  continue
        if hash.has_key(substr):
            hash[substr] += 1
        else:
            hash[substr] = 1
    return(hash)
```

前回課題の解答例(Cont.)

```
if __name__ == '__main__':
    import sys
    str = sys.argv[1]
    d = countsubstrings(str)
    list = [(d[k], k) for k in d.keys()]
    list.sort()
    list.reverse()
    for (v, k) in list:
        print k, " occurs ", v, " times"
```