Python勉強会@HACHINONE

第4章

関数、スコープ、抽象化

お知らせ

Python勉強会@HACHINOHEでは、ジョン・V・グッターグ『Python言語による
プログラミングイントロダクション』近代科学社、2014年をみんなで勉強しています。

この本は自分で読んで考えて調べると力が付くように書かれています。

自分で読んで考えて調べる前に、このスライドを見るのは、いわばネタバレを
聞かされるようなものでもったいないです。

是非、本を読んでからご覧ください。

チューリング完全

Python勉強会@HACHINOHE

- ここまでで紹介した内容で、チューリング完全
  - 数字、代入、入出力、比較、ループなど
- なので、これだけでコンピュータでできることは何でもできる

- できましたよね？

- ただし、現実に使用する上では、いくつか課題も
  - 再利用性とか

関数

Python勉強会@HACHINOHE

- コード列に名前をつけて呼び出せるようにしたもの
  - 関数の定義

```
def 関数名(仮引数の並び):
    関数本体
```

```
def max(x, y):
    if x > y:
        return x
    else:
        return y
```

  - 関数の呼び出し

```
関数名(実引数の並び)
```

```
max(3, 4)
```

- 関数を呼び出すと、実行箇所は関数本体に移動し、returnか最後まで行くと呼び出したところに戻る
- returnで値を返せる。returnがなければNoneが返る

指練習: 4.1.1節

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
def isIn(s1, s2):
    """s1とs2を文字列と仮定
        s1とs2のどちらかにもう片方が含まれればTrue、
        含まれなければFalseを返す。"""

    # 長い文字列をlong、短い文字列をshortに代入
    if len(s1) > len(s2):
        long, short = s1, s2
    else:
        long, short = s2, s1

    # 長い文字列を、前から順番に短い文字列と同じ長さに切り出し、
    # 短い文字列と比較
    for i in range(len(long) - len(short) + 1):
       if long[i:i + len(short)] == short:
            return True
    return False
```

指練習: 4.1.1節

Python勉強会@HACHINOHE

え！？ strのin演算子を使っていいよ？

```
# -*- coding: utf-8 -*-
def isIn(s1, s2):
    """s1とs2を文字列と仮定
       s1とs2のどちらかにもう片方が含まれればTrue、
       含まれなければFalseを返す。"""

    # 長い文字列をlong、短い文字列をshortに代入
    if len(s1) > len(s2):
        long, short = s1, s2
    else:
        long, short = s2, s1

    if short in long:
        return True
    else:
        return False
```

指練習: 4.1.1節

Python勉強会@HACHINOHE

strのin演算子を使えば実質1行でも書けるよ

from 参加者

```
# -*- coding: utf-8 -*-
def isIn(s1, s2):
    """s1とs2を文字列と仮定
        s1とs2のどちらかにもう片方が含まれればTrue、
        含まれなければFalseを返す。"""
    return (s1 in s2) or (s2 in s1)
```

指練習のテスト: 4.1.1節

Python勉強会@HACHINOHE

```
def testIsIn():
    # 空文字列は非空文字列に含まれていると見なす
    print isIn('abcdefg', '') == True
    print isIn('', 'abcdefg') == True

    # 空文字列同士
    print isIn('', '') == True

    # 完全一致
    print isIn('abcdefg', 'abcdefg') == True

    # 部分一致とその境界チェック
    # 最初
    print isIn('abcdefg', 'abc') == True
    print isIn('abcdefg', 'aab') == False
    print isIn('abcdefg', 'abb') == False

    # 中間
    print isIn('abcdefg', 'bcd') == True
    print isIn('abcdefg', 'cbc') == False
    print isIn('abcdefg', 'bcc') == False

    # 末尾
    print isIn('abcdefg', 'efg') == True
    print isIn('abcdefg', 'fgg') == False
    print isIn('abcdefg', 'ffg') == False
```

キーワード引数とデフォルト値

Python勉強会@HACHINOHE

- 引数は、位置指定引数(書いた順番)だけでなく、キーワードでも指定でき、デフォルト値を指定するときに便利
  - 関数の定義

```
def printName(firstName, lastName, reverse = False):
    if reverse:
        print lastName + ', ' + firstName
    else:
        print firstName, lastName
```

  - 関数の呼び出し

```
printName('Olga', 'Puchmajerova') # Olga Puchmajerova
printName('Olga', 'Puchmajerova', True) # Puchmajerova, Olga
printName('Olga', 'Puchmajerova', reverse = True) # Puchmajerova, Olga
```

変数の有効範囲スコープ

Python勉強会@HACHINOHE

- 関数内では、関数が宣言されたスコープで有効な変数、仮引数、関数内で定義された局所変数が有効
  - 局所変数は関数外からアクセスできない
  - 実引数の値は仮引数に代入される
  - スタック(積み上げ、後入れ先出し)フレームというしくみで実現

関数の内部関数、戻り値としての関数

Python勉強会@HACHINOHE

```
# 関数fの宣言
def f(x):

    def g():
        # 局所変数x
        x = 'abc'
        print 'x =', x

    def h():
        # 外部のxを参照
        z = x
        print 'z =', z

    x = x + 1
    print 'x =', x
    h()
    g()
    print 'x =', x
    return g

# メイン・プログラム
x = 3
z = f(x) # f(3)が評価され、zにはf内のgが代入
print 'x =', x
print 'z =', z
z() # zはgなので、g()が評価される
```

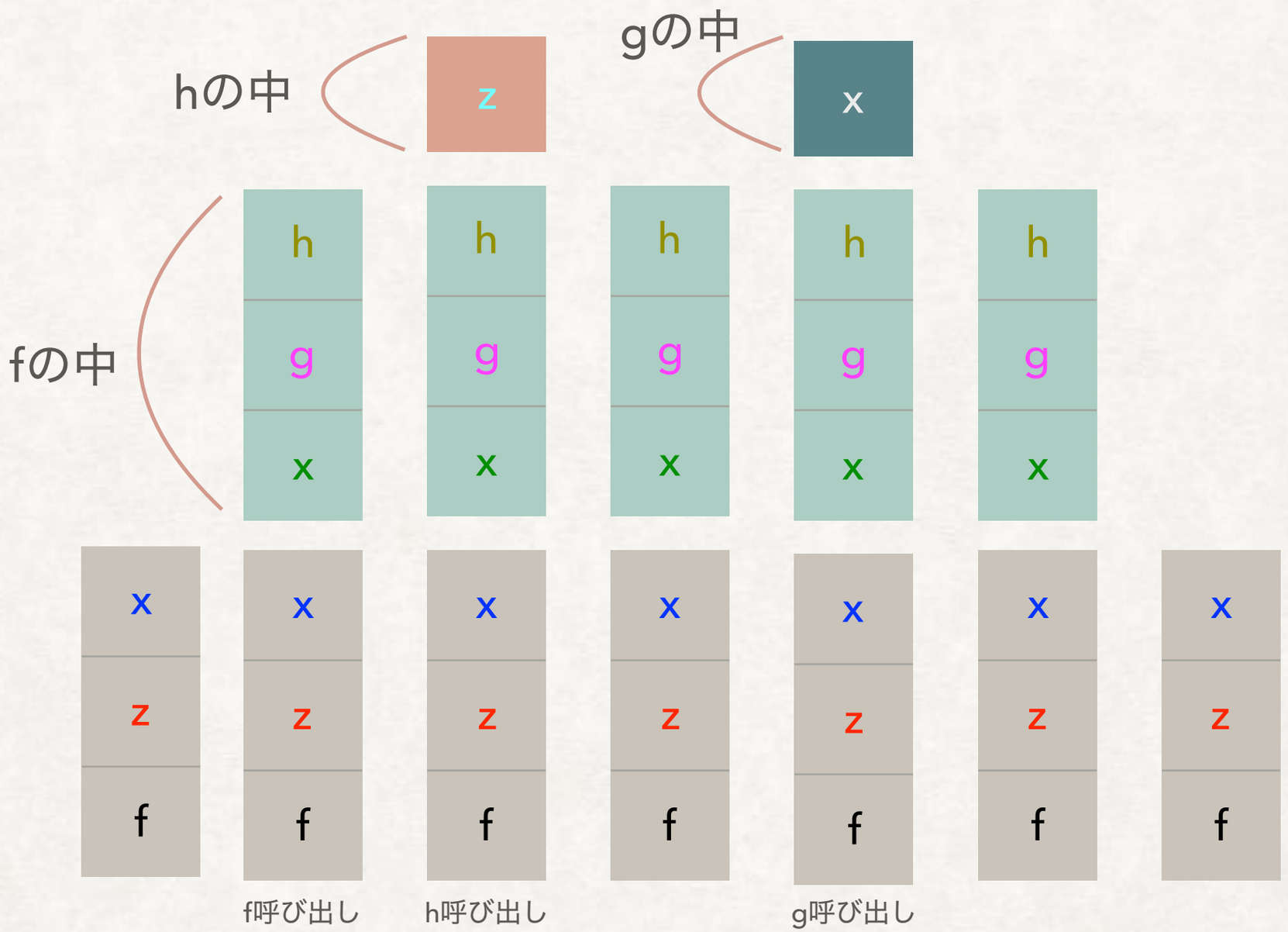
実行

```
# f(3)の評価はじまり
  x = 4

  # h()の評価はじまり
    z = 4
  # h()の評価おわり

  # g()の評価はじまり
    x = abc
  # g()の評価おわり

  x = 4
# f(3)の評価おわり

x = 3
z = <function g at 0x105ce8050>

# z()、すなわちg()の評価はじまり
  x = abc
# g()の評価おわり
```

スタックフレーム

Python勉強会@HACHINOHE

```
def f(x):
    def g():
        x = 'abc'
        print 'x =', x
    def h():
        z = x
        print 'z =', z
    x = x + 1
    print 'x =', x
    h()
    g()
    print 'x =', x
    return g

x = 3
z = f(x)
print 'x =', x
print 'z =', z
z()
```



実行結果

Python勉強会@HACHINOHE

```
def f(x):
   def g():
        x = 'abc'
        print 'x =', x
    def h():
        z = x
        print 'z =', z
    x = x + 1
    print 'x =', x
    h()
    g()
    print 'x =', x
    return g

x = 3
z = f(x)
print 'x =', x
print 'z =', z
z()
```

```
x = 4 # f()
z = 4 # h()
x = abc # g()
x = 4 # f()
x = 3 # 外
z = <function g at 0x10676d500> # 外
    # zにはf(x)が代入され、f(x)はgを返している
x = abc # zを関数として実行、つまりg()
```

仕様

Python勉強会@HACHINOHE

- ドキュメンテーション
  - 利用者との契約: 仮定(引数の仕様)と保証(実行結果の仕様)を記述
  - docstringで記述: """と"""で囲う
  - 具体的な記述方法はいろいろ: As you like!
- テスト
  - その関数のテストを記述
  - テストのためのライブラリもいろいろ
  - assertは後で

仕様を作る上でのポイント

Python勉強会@HACHINOHE

- うまく分解する
  - 問題を合理的、本質的に分解する
  - それにより再利用性が高まる
- うまく抽象化する
  - コードを書く際に、コードを読む範囲を少なくしたい
  - 関数を利用して、仕様だけを共有し、詳細な実装は知らなくても使えるようにする

再帰

Python勉強会@HACHINOHE

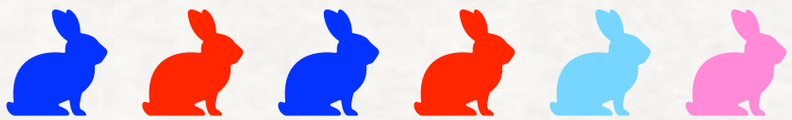
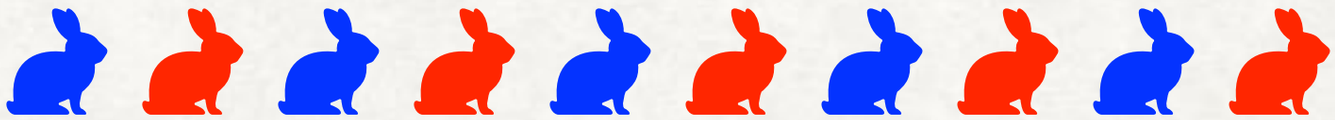
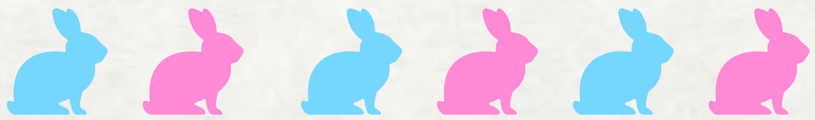
- 帰納的定義
  - 基部(初項など)と漸化式
- 例: 階乗の計算

```
# 繰り返しで階乗を計算
# n! = n * (n -1) * (n - 2) * ... * 1
def factI(n):
    """n > 0を整数と仮定
       n!を返す"""
    result = 1
    while n > 1:
        result = result * n
        n -= 1
    return result
```

```
# 再帰で階乗を計算
# n! = n * (n -1)!、1! = 1
def factR(n):
    """n > 0を整数と仮定
       n!を返す"""
    if n == 1:
        return n
    else:
        return n * factR(n - 1)
```

フィボナッチ数列

Python勉強会@HACHINOHE

- フィボナッチ(ピサのレオナルド)がアラビア数学を紹介した『算盤の書』1202年に掲載。インドの数学者ピンガラによる
  - 0ヶ月目の終わり:1 
  - 1ヶ月目の終わり:1 
  - 2ヶ月目の終わり:2 
  - 3ヶ月目の終わり:3 
  - 4ヶ月目の終わり:5 
  - 5ヶ月目の終わり:8 
  

子ウサギは1ヶ月で親ウサギになる

親ウサギの番は1ヶ月で子ウサギの番を産む

フィボナッチ数列の式

Python勉強会@HACHINOHE

- 漸化式

```
メス(0) = 1
メス(1) = 1
メス(n) = メス(n-1) + メス(n-2)
```

- コード

```
# -*- coding: utf-8 -*-
def fib(n):
    """n >= 0を整数と仮定
       n番目のフィボナッチ数を返す"""
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

def testFib(n):
    for i in range(n+1):
        print str(i) + '番目のフィボナッチ数は' + str(fib(i))
```

指練習: 4.3.1節

Python勉強会@HACHINOHE

```
fib(5) = fib(4) + fib(3)
       = (fib(3) + fib(2)) + (fib(2) + fib(1))
       = (fib(2) + fib(1)) + fib(2) + fib(2) + fib(1)
```

合計3回

回文

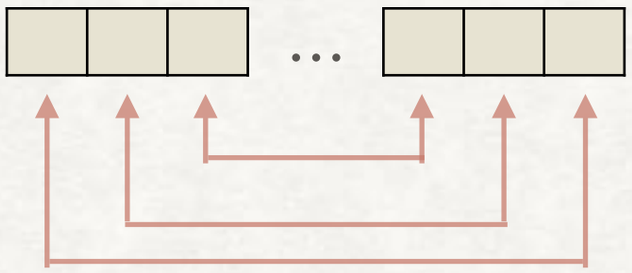
Python勉強会@HACHINOHE

- 前から読んでも後ろから読んでも同じ文
  - "Able was I ere I saw Elba"
    「エルバ島を見るまでわたしに不可能はなかった」
  - 「長き夜の 遠の睡りの 皆目醒め 波乗り船の 音の良きかな」

回文判定のコード

Python勉強会@HACHINOHE

- 補助関数の活用
  - 小文字だけにするtoChars
  - 外側から内側へ向かい一致するかを調べていくisPal



- メソッド呼び出し
  - オブジェクト.メソッド()
- 短絡評価
  - andは左から順番に評価していき、Falseになったら打ち切り

```
# -*- coding: utf-8 -*-
def isPalindrome(s):
    """sを文字列と仮定
       sが回文ならTrueを返し、それ以外ならFalseを返す
       ただし句読点、空白を無視し、大文字・小文字を区別しない"""
    def toChars(s):
        s = s.lower()
        letters = ''
        for c in s:
            if c in 'abcdefghijklmnopqrstuvwxyz':
                letters = letters + c
        return letters

    def isPal(s):
        if len(s) <= 1:
            return True
        else:
            return s[0] == s[-1] and isPal(s[1:-1])

    return isPal(toChars(s))
```

分割統治

Python勉強会@HACHINOHE

- 問題を、より簡単な部分問題に分けて解き、結果を結合して解決

- 回文の例では、以下の2つに分けて解き、andで結合

1.同じ問題のより簡単なバージョン

  - 左右を1文字ずつ削った文字列もまた回文か？

2.やり方がわかっている単純な方法

  - 両端の文字は等しいか？

テスト付き回文判定のコード

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
def isPalindrome(s):
    """sを文字列と仮定
       sが回文ならTrueを返し、それ以外ならFalseを返す
       ただし句読点、空白を無視し、大文字・小文字を区別しない"""
    def toChars(s):
        s = s.lower()
        letters = ''
        for c in s:
            if c in 'abcdefghijklmnopqrstuvwxyz':
                letters = letters + c
        return letters

    def isPal(s):
        print '   isPalが' + s + 'を引数に呼び出されました'
        if len(s) <= 1:
            print '   基底状態に対しTrueが返されました'
            return True
        else:
            answer = s[0] == s[-1] and isPal(s[1:-1])
            print '   ' + s + 'に対し' + str(answer) + 'が返されました'
            return answer

    return isPal(toChars(s))

def testIsPalindrome():
    print 'Try dogGod'
    print isPalindrome('dogGod')
    print 'Try doGood'
    print isPalindrome('doGood')
```

広域(グローバル)変数

Python勉強会@HACHINOHE

- 広域変数はプログラムの中全体からアクセスできる
  - 本当に必要な場合にだけ使用すべき
  - 変数の値がプログラムのあちこちで操作されると理解困難に
- 広域変数の宣言

```
global 広域変数名
```

fib関数の呼び出し回数を表示する

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
def fib(x):
    """ x >= 0 を整数と仮定
        x番目のフィボナッチ数を返す"""
    global numFibCalls
    numFibCalls += 1
    if x == 0 or x == 1:
        return 1
    else:
        return fib(x - 1) + fib(x - 2)

def testFib(n):
    for i in range(n + 1):
        global numFibCalls
        numFibCalls = 0
        print str(i) + '番目のフィボナッチ数=' + str(fib(i))
        print 'fib関数は' + str(numFibCalls) + '回呼び出されました。'
```

モジュール

Python勉強会@HACHINOHE

- プログラムを複数のファイルに分割
- モジュールの読み込み

```
import モジュール名
```

- モジュール内のオブジェクトなどにアクセス

```
モジュール名.オブジェクト名など
```

- モジュールを現在の名前空間に読み込み

```
from モジュール名 import *
```

- モジュールを再読み込み

```
reload()
```

ファイル入出力

Python勉強会@HACHINOHE

- ファイルハンドルを通じてファイル入出力を行う
- open(ファイル名, モード): ファイルを開きファイルハンドルを返す
  - モード: 'w' 書き込み、'r' 読み込み、'a' 追加書き込み
- ファイルハンドル.close(): ファイルを閉じる
- ファイルハンドル.読み込みメソッド(): ファイルから読み込む
  - 読み込みメソッド: 文字列read、1行readline、全行readlines
- ファイルハンドル.書き込みメソッド(): ファイルに書き込む
  - 書き込みメソッド: write(文字列)、writelines(文字列シーケンス)

ファイル操作のコード例

Python勉強会@HACHINOHE

```
# -*- coding: utf-8 -*-
nameHandle = open('kids', 'w') # kidsを書き込みモードで開く
nameHandle.write('Michael\n')  # 'Michael改行'を書き込む。
nameHandle.write('Mark\n')     # 'Mark改行'を書き込む
nameHandle.close()             # kidsを閉じる
nameHandle = open('kids', 'r') # kidsを読み込みモードで開く
for line in nameHandle":       # 1行ずつlineに取り出す
    print line[:-1]            # 最後の改行を除いて出力
nameHandle.close()             # kidsを閉じる
```

まとめ

Python勉強会@HACHINOHE

- 実用的なプログラムを書くために
  - 個々の具体的なテクニックよりも、しくみや考え方を理解
- 関数: プログラムを抽象化
- スコープ: スタックフレームというしくみ
- 仕様: 仮定と保証を記述
- 再帰: プログラムを帰納的に書くこともできる
- 広域変数
- モジュール: プログラムの分割
- ファイル入出力