2018年航空宇宙情報システム学第2
第2部「プログラミングと数値計算」

第3回 Python入門 2
～条件文・再帰呼び出し・繰り返し文～

2018年5月22日

5章 条件文と再帰呼び出し

(4章 Case Study は割愛します)

切捨除算と剰余演算

- **演算子 // は切り捨て除算を行う**

```
>>> minutes = 105
>>> hours = minutes // 60
>>> hours
1
```

- **一方、剰余演算子 % は、整数同士の割り算の「余り」を求める**

```
>>> remainder = minutes % 60
>>> remainder
45
```

- **整数 x の下n桁の数字を知るには、**

```
>>> x % (10**n)
```

論理式

- **論理式**（Boolean expression）：**真**(True)**あるいは偽**(False)**のいずれかの値を取る式**

```
>>> type (True)
<class 'bool'>
```

- **関係演算子**：**左辺と右辺を比べる論理式を作る**

  例えば、x = 5 , y = 3 **の場合**

```
>> x == y      # x と yは等しい→偽
False
>>> x >= y     # x は y 以上→ 真
True
>>> x != y     # x と y は異なる→真
True
```

論理演算子

- **数学、論理学でもおなじみ（？）**
- **3種類の論理演算子** :
  - **論理積** : and
    （**例**） >>> x > 5 and x < 10
  - **論理和** : or
    （**例**） >>> x % 2 == 0 or x % 3 == 0 # x **は**2**か**3**の倍数**
  - **否定** : not
    （**例**） >>> not (x < 5) # x **は**5**未満ではない**
  - （**注意**） Python**では非ゼロの数値は真とみなされる**
    （**例**） >>> -1 and True # **負の値でも**True
    True

条件実行

- **条件文：ある条件が成り立つときに実行**
  - **プログラミングでは必須**
- **最もシンプルな条件文　(if 文)**

```
>>> if x > 0:                          # ヘッダー
...     print('x is positive')         # 本文 (ボディ)
...
x is positive
```

  - **ヘッダーは : (コロン) で終わる**
  - **本文はインデントする。複数行あっても良い**

選択実行

- **選択実行** (alternative execution) : **いわゆる、** if – else **文**
- **簡単な例** （x = 5 **とする**)

```
>>> if x % 2 == 0:
...     print('x is even‘)
... else:
...     print('x is odd‘)
...
x is odd
```

連鎖条件文

- if – elif – else **という順番で条件実行**
- **例**：x = 5 **とする**

```
>>> if x < 0:
...     print('x is negative‘)
... elif x > 0:
...     print('x is positive‘)
... else:
...     print('x equals to zero‘)
...
x is positive
```

- elif **文は**2**個以上あっても良い**

入れ子条件文

- **条件文の中に別の条件文が入っていても良い**
- **例** (x = 5, y = 3 **とする**)

```
>>> if x == y:
...     print('x equals to y‘)
... else:
...     if x > y:
...         print('x is larger than y‘)
...     else:
...         print('x is smaller than y‘)
...
x is larger than y
```

- **インデントが深く**なっている点に注目

再帰呼び出し

- **関数が他の関数を呼び出すのは普通だが、自分自身を呼び出すこと（再帰呼び出し）もある**

```
def countdown(n):
    if n <= 0:
        print('Blastoff!‘)
    else:
        print(n)
        countdown(n-1)    # 引数を変えて再帰呼び出し
```

  - **この場合はFor（ループ）文で書いた方が早いが、再帰呼び出しの方が「嬉しい」場合もある（後述）**
  - **無限の再帰呼び出しを行うとエラーが出て止まる**
  - **再起呼び出しの典型例: 階乗の計算（後述）**

キーボード入力

- **組み込み関数** input() **を使ってユーザーのキーボード入力を扱うことができる**

  >>> name = input(‘What is your name ? ¥n')

  What is your name ? ← **引数で与えた文字列が表示**

  Taro ← **ユーザーの入力**

  >>> print(name+ ‘, nice to meet you.‘)

  Taro, nice to meet you.

  – ‘¥n’ **は改行文字** (¥ **は本当は半角バックスラッシュ**)

  – **ユーザーが入力した文字列が**name**に代入される**

- **対話型プログラムを作るのに有用**

デバッグの話

- (行始めの)空白文字に気をつけよう

```
>>>  y=10          ← y の前にスペースが1個入っている
  File "<stdin>", line 1
    y=10
    ^
IndentationError: unexpected indent
```

- エラーが見つかる場所と真の原因がある場所は必ずしも同じではない

```
import math
signal_power = 9
noise_power = 10
ratio = signal_power // noise_power   ← 真の原因がある場所
decibels = 10 * math.log10(ratio)     ← 見つかる場所
```

6章 出力のある関数
(Fruitful Functions)

戻り値

- 今まで自作した関数は戻り値が無いものだった（void 関数と呼ばれる）
- 戻り値を返すには、return文を用いる

```
def area(radius):
    a = math.pi * radius**2
    return a
```

  - この例では、a が一時変数の役割をしている
  - 一時変数を用いずに書くこともできる

```
def area(radius):
    return math.pi * radius**2
```

  - コンパクトだが、デバッグには向かない

注意：この関数を使用する前に、
>> import math
(mathモジュールのインポート)
を実行しておく必要がある。

条件分けと戻り値

- 条件を分けて戻り値を返す場合、あらゆる場合で何らかの値を返すように書かないといけない
  - (例) 絶対値を返す関数

ダメなコード

```
def absolute_value(x):
    if x < 0:
        return -x
    if x > 0:
        return x
```

x = 0 の場合、戻り値が無い (None)

良いコード

```
def absolute_value(x):
    if x < 0:
        return -x
    else:
        return x
```

(注) 組み込み関数 abs を使えば良いので実用的な意味は無い。

段階的なコーディング

複雑・大規模な関数を書く場合、デバッグしながら少しずつ書いていくと良い　(例：2点間の距離を求める関数)

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ver.0.1　**構文的には正しいミニマルプログラム**

```
def distance(x1, y1, x2, y2):
    return 0.0
```

Ver.0.2　**途中計算を表示**

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    print('dx is', dx)
    print('dy is', dy)
    return 0.0
```

Ver.0.3　**途中計算を表示**

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    print('dsquared is: ', dsquared)
    return 0.0
```

Ver.1.0　**(完成版)**

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    result = math.sqrt(dsquared)
    return result
```

scaffolding：(**建設現場の**) “**足場**”

関数の合成

- **既出の自作関数** area() **と** distance() **を使って、「中心点と円周上の点の座標を与えられたときに、この円の面積を求める」関数を作ってみる**

```
def circle_area(xc, yc, xp, yp):
    radius = distance(xc, yc, xp, yp)
    result = area(radius)
    return result
```

一時変数を使う書き方：
長たらしいが、理解しやすい
（デバッグ向き）

```
def circle_area(xc, yc, xp, yp):
    return area(distance(xc, yc, xp, yp))
```

関数の合成を使う書き方：
コンパクトだが、やり過ぎると
後で理解しにくくなる

2値関数

- 2値 (Boolean) 関数 : True または False を返す
  - 他の関数の中で、条件をチェックするのに有用
  - (例) x が y で割り切れるかどうかをチェックする関数

素直に書くと。。

```
def is_divisible(x, y):
    if x % y == 0:
        return True
    else:
        return False
```

実は、1行で書けてしまう

```
def is_divisible(x, y):
    return x % y == 0
```

使い方の例

```
if is_divisible(x, y):
    print('x is divisible by y‘)
```

再帰呼び出し再び

- **今まで学んだPythonのテクニックで、「計算可能な」関数は全て実装できる**
  - **ポイントは、再帰呼び出し** (recursion)
  - **理論的には、チューリングのテーゼに依る**
- **(例) n の階乗を計算する関数**

```
def factorial(n):
    if n == 0:
        return 1
    else:
        recurse = factorial(n-1)
        result = n * recurse
        return result
```

注意：一見、これで良さそうだが、このプログラムには実用上の問題がある (後述)

再帰呼び出しの例：Fibonacci関数

- fibonacci関数を以下のように定義

$$\text{fibonacci}(n) = \begin{cases} 0 & (\text{if } n = 0) \\ 1 & (\text{if } n = 1) \\ \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & (\text{otherwise}) \end{cases}$$

- Python で実装すると

```
def fibonacci (n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

上の数学定義との類似性に着目

注意：このプログラムにも同様の実用上の問題がある。

型のチェック

- 既出の factorial(n) 関数の欠陥 : 引数に浮動小数や負の値を与えると無限ループになる
- 対策1 : 関数を一般化する (ガンマ関数)
  - しかし、負の値には対応せず。→ ここでは割愛
- 対策2 : 引数の型 (と範囲) をチェックする

```
def factorial (n):
    if not isinstance(n, int):   # 整数型(int)以外を排除
        print('Factorial is only defined for integers.‘)
        return None
    elif n < 0:        # 負の値を排除
        print('Factorial is not defined for negative integers.‘)
        return None
    elif n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

- プログラムは長くなるが、バグ発生の可能性は減る

7章 繰り返し文

再代入

- Python (**や多くのプログラミング言語で**) **= 記号は等号ではなく変数への値の代入を意味する**
- **同じ変数に複数回、値を代入することも可能**

```
>> x = 5
>> x
5
>> x = 7          #新しい値
>> x
7
>> a = 5
>> b = a    # 変数 b に 変数 a の値(5)を代入
>> a = 3    # 変数 a に新しい値を代入。bは変化しない
```

変数の更新

- **複数回代入の典型的な例**:

```
>>> x = x + 1
```

  - **変数** x **の現在の値を取り出し、それに**1**足したものを** x **に代入する**
  - **ただし、変数** x **は既に存在していないといけない**

```
>> x = 0     # 変数 x の「初期化」
>> x = x + 1
```

  - **変数の値を**1**増やすことをインクリメント、逆に**1**減らすことをデクリメントと呼ぶ**

```
>> x = x - 1     # デクリメント
```

while文による繰り返し

- **以前、再起呼び出しを用いて作成した countdown 関数を、while文で書き直すと、**

```
def countdown(n):
    while n > 0:   # n>0である限り下のブロックを繰返す
        print(n)    # 現在の nの値を表示
        n = n-1    # nの値を1つ減らす
    print 'Blastoff! ’
```

- **この場合、nが整数か実数であれば、必ず終了 (無限ループにならない)**

(例) コラッツの問題

- コラッツの予想：「nを任意の自然数とする。nが偶数なら2で割り、奇数なら3を掛けて1を足すという操作を繰返すと、最終的に1に到達する。」
  - 未証明だが反例は見つかっていない

```
def sequence(n):
    while n != 1:
        print(n)
        if n % 2 == 0:    # n is even
            n = n/2
        else:             # n is odd
            n = n*3+1
```

引数 n に負の値を与えると無限ループになるので注意

http://en.wikipedia.org/wiki/Collatz_conjecture

break 文

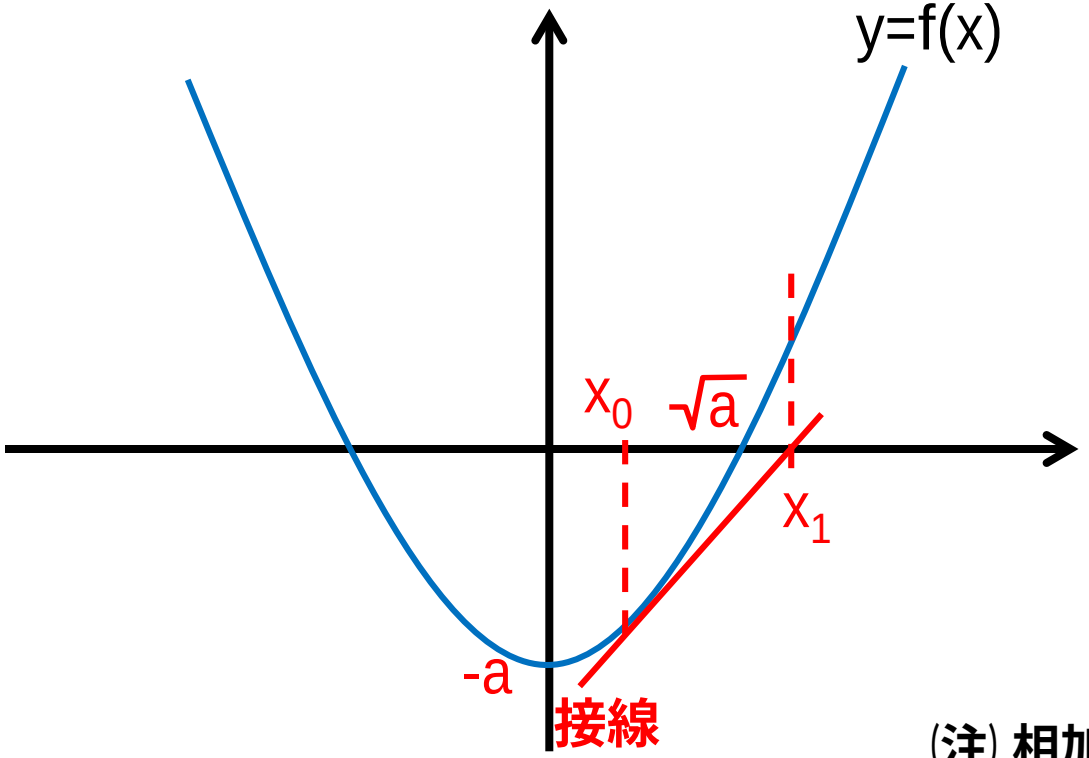
- ループ (繰り返し) から強制的に抜け出る手段
- 例: 入力された文字が'done'に一致するまでプロンプトを出し続けるプログラム

```
while True:   # 無限ループ
    line = input('> ')
    if line == 'done':
        break
    print(line)
```

- 無限ループ + break は良く使われるテクニックの1つ

⑴例⑴ 平方根を反復法で求める

- 正の実数 a の平方根を**ニュートン法**で求める
- (数学的な)関数 $f(x) = x^2 - a$ を考えたとき、$f(x)=0$ の解を求めるということ



同様に、$x_2, x_3, \ldots x_\infty$を計算していくと、a の平方根に収束する

$$x_{n+1} = \frac{x_n + \frac{a}{x_n}}{2}$$

⑴注⑴ 相加相乗平均の応用と見ることもできる

平方根を求める（続き）

```
>>> a = 5
>>> x = 2.0 #初期値
>>> while True:
...     print(x)
...     y = (x + a/x) / 2.0
...     if y == x:
...         break
...     x = y
...
2.0
2.25
2.23611111111
2.23606797792
```

通常は、ある程度の誤差を許容して収束判定を行う。

```
if abs(y-x) < 1.0e-6 :
    break
```

関数にすると、

```
def mysqrt(a):
    x = 2.0
    while True:
        y = (x+a/x) / 2.0
        if abs(y-x) < 1.0e-6:
            break
        x = y
    return x
```

“アルゴリズム”とは？

- ニュートン法は「アルゴリズム」の例
- アルゴリズムとは、「ある種類の問題を解くための機械的な手続き」
- アルゴリズムの例: 9×n (九九の九の段)
  - 9, 18, 27, …, 81 を暗記するのではなく、10の位の数字が0, 1, 2, …と1ずつ増えていき、1の位が9,8,7,… と1ずつ減っていくという規則性に着目する
  - つまり、10の位は n-1、1の位は10-n として求める

今日の宿題

- 2つの整数 a, b を引数として取り、a が b のべき乗であるかどうかを判定する関数 is_power(a,b) を作成せよ
  - 再帰呼び出しを使うことを想定しているが、無理して使う必要は無い
    (余裕があれば、再起呼び出しを使うバージョン、使わないバージョンを作って比較してみよう。)
- プログラムファイルに homework02.py という名前を付けて、ITC-LMSから提出すること
- 締切：5月29日午前8時 (次回授業の直前)