

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2回 Python入門 1 ～ 変数と関数 ～

2018年5月15日

## 2 章 变数, 式, 文

# 変数への代入

- 代入文(assignment statement)は、新しい**変数**を生成し、それに**値**を与える

```
>>> message = 'And now for something completely different'
```

```
>>> n = 17
```

```
>>> pi = 3.141592
```

- 記号 **=** は、「**等号**」ではなく、「**代入**」を表す。

- 比較演算子の等号には**==**を使う

- **変数の型** = その変数に代入された**値の型**

```
>>> type(pi)
```

- C/C++などと異なり、変数の型宣言は必要ない

# 変数名とキーワード

- 変数名の付け方のルール
  - 名前の長さには制限無し(どれだけ長くても良い)
  - 文字と数字を含んで良いが**数字で始まる名はNG**
  - 大文字も使用可だが**小文字のみを使うのが慣習**  
また、**大文字と小文字は区別される** (case sensitive)
  - アンダスコア\_以外の記号(+,#,\$など)は使用不可
- キーワード(予約語)は変数名に使えない
  - Python3 には33個のキーワードがある
  - (例) and, as, break, class, if, for, try, lambda など
  - 一部にキーワードを含むのは可。(例) a\_and\_b
- 通常は、プログラマにとって意味のある(便利な)名前を付ける

# 式と文

- 式 (expression): 値、変数、演算子の組み合わせ

>>> 42  
42  
>>> n  
17

いずれも「式」

(対話モードでは) 式を評価して  
生じた値が出力される

– 式は、評価された結果、**値を生じる**

- 文 (statement): 変数を生成したり、値を表示するなど、プログラムの最小実行単位

>>> n = 17  
>>> print(n)  
**17**

これは表示しただけであって、値を出力したのではない！

– 文は必ずしも値を持たない

# スクリプトモード

- 今、使っている対話モードでは、  
    >>> miles = 26.2  
    >>> miles \* 1.61  
42.182      ← 計算 (式の評価) 結果が表示される
- 以下のスクリプト (test01.py) をエディタで作成  
    miles = 26.2  
    miles \* 1.61
- “F5” キーを押して実行すると.. 結果が表示されない      対話モードとの挙動の違い
- スクリプトモードで計算結果を表示させるには、  
    miles = 26.2  
    print(miles \* 1.61)

# 演算の順序

- 基本的には、数学で習った通りの順番
- 演算の優先順位 (高い順)

1. 括弧: ( )
2. べき乗関数: \*\*
3. 掛け算、割り算: \* , /
4. 足し算、引き算

```
>>> 2**(3+1) + 8 / 2  
20
```

- 優先順位が同じ演算は、左から右へ  
(ただし、**冪演算\*\* は例外**)

```
>>> 2**3**2  
512
```

← この場合は、 $2^{3^2}$  と解釈される

# 文字列の演算

- 文字列に対して数学的な演算は不可
  - '2'-'1' 'eggs'/'easy' 'third'\*'a charm'  
は全てNG
  - '2' は文字列であって数字ではない点に注意
- ただし、+ と \* は連結と反復に使うことができる
  - 文字列 + 文字列 は連結になる
    - >>> 'break' + 'fast' は 'breakfast'
    - >>> 'pen' + 'pineapple'
    - 'penpineapple'
  - 文字列 \* 整数 は反復になる
    - (例) >>> 'Spam' \* 3 は 'SpamSpamSpam' になる
- 日本語でもやってみよう



# コメント文

- 主にスクリプトモードで有用
- プログラムで説明やメモなどを加えたい場合に使う
- Python では、**#** (シャープ) で始まる文はコメントとみなされる
- (例) 独立したコメント行  
**# compute the percentage of the hour that has elapsed**  
percentage = (minute \* 100) / 60.0
- (例) 行の末尾にコメントを加える場合  
percentage = (minute \* 100) / 60.0 **# percentage of an hour**
- 日本語を使う場合は文字コードを指定する必要あり

# デバッキング

- バグには3種類ある

1. 構文エラー (Syntax error)

>>> 1+2)      # (1+2) と書くつもりだった

>>> 1+^3\$    #意味をなさない無効な構文

2. 実行時エラー (Runtime error)

>>> 4/0.0    #ゼロによる割り算

3. 意味エラー (semantic error)

「1+3 と書くつもりで 1\*3 と書いてしまった」、  
「0の数が1つ多かった」、など

- 発見して解決する難易度は、一般に、  
構文エラー < 実行時エラー < 意味エラー

# 3章 関数

# 関数の呼び出し

- 関数 (function) : あるタスクを行うために複数の文をまとめたもの
  - プログラミング言語によって、サブルーチン、手続き、メソッド、マクロ、などとも呼ばれる
  - 通常、関数は「名前」によって呼び出される
- ```
>>> type(32)  
<type 'int'>
```

  - type が関数の名前
  - 括弧の中 (関数への入力) を引数 (argument) と呼ぶ
  - 関数の出力を戻り値 (return value) と呼ぶ

# 型変換関数

- 型を変換する組み込み関数たち (int, float, str)

```
>>> int('32')    # 文字列'32'を整数型に変換  
32
```

```
>>> int('Hello') # これは無理  
ValueError: invalid literal for int(): Hello エラー
```

```
>>> float(32)    ## 整数を浮動小数型に変換  
32.0
```

```
>>> str(3.14159) #浮動小数型を文字列に変換  
'3.14159'
```

- int(3.9) や int(-2.3) はどうなる？

# 数学関数

- 主な数学的関数は **math モジュール**に含まれる
  - モジュールとは、**複数の関数をまとめたもの**
  - モジュールを読み込むには、**import 文**を使う
  - モジュールに含まれている関数を呼び出すには、**モジュール名.関数名(引数)**とする
  - `from モジュール名 import 関数名` という方式は後述
- **math モジュールと関数の利用例**
  - `>>> import math`      ← 1回呼び出せば良い
  - `>>> radians = 0.7`
  - `>>> height = math.sin(radians)`
- `help(math)`で含まれる関数、定数を調べられる

# 変数、式、関数の合成

- プログラミング言語の特徴: 小さな部品を組み合わせて大きなプログラムを作ることができる

```
>>> x = math.sin(degrees / 360.0 * 2 * math.pi)
```

変数や演算子などを含む式を引数としている

```
>>> x = math.exp(math.log(x+1))
```

関数の戻り値を別の関数の引数にしている

- ただし、**代入文の左辺**は(1つの)変数名でなければならない

```
>>> minutes = hours * 60      ← これはOK
```

```
>>> hours * 60 = minutes      ← これはダメ
```

注: タプル(後述)を使うと、複数の変数に同時に代入することも可

# 新しい関数の追加

- 新しい関数の定義: 関数の名前と実行する内容 (一連の文) を記述する

関数定義の例

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print("I sleep all night and I work all day.")
```

- 関数名: `print_lyrics`
- 括弧の中が空 → 引数を取らないことを意味する
- 先頭行をヘッダー、残りを本文 (body) と呼ぶ
- ヘッダーは **コロン (:) で終わる**
- 本文は (4文字分) **インデント (字下げ) する**
  - 一貫していれば2字でも8字でも構わないが4字を推奨



# 関数の作成 ( 続き )

- 対話モードでも関数の定義は可能

```
>>> def print_lyrics():  
... print("I'm a lumberjack, and I'm okay. ")  
... print("I sleep all night and I work all day. ")  
...     ←対話モードで関数定義を終了させるための空行
```

ネタ元:  
'Monty Python'  
の挿入歌  
"Lumberjack Song"

- 呼び出し方は組み込み関数と同じ

```
>>> print_lyrics()  
I'm a lumberjack, and I'm okay.  
I sleep all night and I work all day.
```

- print\_lyrics を2回実行するrepeat\_lyrics関数を定義してみよう

# パラメータと引数

- 関数に与えられた引数は、パラメーターと呼ばれる変数に代入される

```
def print_twice(bruce):    # 引数がパラメーターbruceに  
    print(bruce)           # 代入される  
    print(bruce)
```

- 関数 `print_twice` に色々な引数を与えて出力を見てみよう。(例: `'Spam', 17, math.pi`)
- 引数は関数が実行される**前に**評価される

```
>>> print_twice('Spam '*4)
```

```
>>> print_twice(math.cos(math.pi))
```

# ローカル変数

- 関数の中で生成された変数はその関数の中でのみ存在する

#2つの引数を結合して2回表示する関数

```
def cat_twice(part1, part2):
```

```
    cat = part1 + part2    # cat はローカル変数
```

```
    print_twice(cat)
```

```
>>> line1 = 'Bing tiddle '
```

```
>>> line2 = 'tiddle bang.'
```

```
>>> cat_twice(line1, line2)
```

- この後、>>> print(cat) としたらどうなるか？
- パラメータもローカル変数→確かめよ

# 出力のある関数とない関数

- 出力(戻り値)のある関数の例: `math.sin` など
  - 戻り値を変数に代入する

```
>>> x = math.sin(0.25*math.pi)
```
  - 対話モードで代入変数を省略すると

```
>>> math.sin(0.25*math.pi)
```

`0.7071067811865475` ←出力(結果)が表示される
  - スクリプトモードでは代入変数を省略すると結果が失われてしまう → 表示させたいときは`print`関数を使う
- 出力のない関数の例: 前述の `print_twice` など  
→ `void` 関数と呼ぶ

```
>>> result = print_twice('Bing')    ←エラーは出ない
>>> print(result)
None
```

# なぜ関数が必要なのか？

大きなプログラムを関数の集合に分割する理由

- 文の集合に(関数の)「名前」を与えることで、プログラムの可読性が高くなる
- 繰り返し部分を関数にすることでプログラムがコンパクトになる。また、変更箇所が少なくなる
- プログラムを関数群に分けることでデバッグの通しが良くなる
- 良く設計された関数は再利用可能

# from を使って関数を import する

- (既出) import によるモジュールの読み込み

```
>>> import math
```

```
>>> math.sin(0.5* math.pi) #モジュール名.オブジェクト名
```
- モジュールからオブジェクト(関数や定数)を直接読み込むこともできる
  - 個別にオブジェクトを読み込む場合

```
>>> from math import exp
```

```
>>> exp(0.5)
```
  - モジュールから**全てのオブジェクト**を読み込む場合

```
>>> from math import *
```
  - 便利だが、**名前(関数名、定数名)の衝突**に注意

# スクリプトモードでのインデント

- Python ではインデント(字下げ)が重要
- 文の固まり(ブロック)を表現するのに利用
  - cf. C言語では{}(中括弧)
- 空白(スペース)文字とタブを混ぜると混乱するので注意
- **スペース4文字**をインデントに使うことを推奨
  - Spyderのエディタもそうになっている
- テキストエディタによっては、タブを(指定個の)スペースに置き換える設定ができる
  - 「Python 編集モード」のあるテキストエディタなら、さらに便利な機能も付属

# 関数に関数を渡す(Exercise 3.2)

- Python では関数オブジェクトを変数に代入したり、他の関数の引数にすることが可能

```
>>> def print_spam(): # 'spam'と表示する関数を作成
...     print('spam')
>>> def do_twice(f): # 関数 f() を2回実行する関数を作成
...     f()
...     f()
>>> do_twice(print_spam) # 引数でprint_spam関数を渡す
```

- エディタで同じ内容をスクリプト(プログラム)に書いて、実行してみよう
- 非常に便利で強力な機能だが、混乱しないために注意が必要



# 今日の宿題

- 教科書 **Exercise 3.3** (アスキー文字で格子模様を描く)を応用して、何か**アスキーアートを表示**するプログラムを作成せよ
  - Exercise 3.3 の解答例は  
<http://greenteapress.com/thinkpython2/code/grid.py>  
にあるので、ダウンロードして実行してみよう  
(Spyder のエディタで開いてF5ボタン)
- プログラムファイルに **homework01.py** という名前を付けて、ITC-LMSから提出すること
- 締切: 5月22日午前8時(次回授業の直前)