クレジット:
UTokyo Online Education Education コンピュータシステム概論 2018 小林克志

東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

Python の基本

この資料は The Python Tutorial (https://docs.python.org/3.6/tutorial/index.html#the-python-tutorial) (日本語版 (https://docs.python.jp/3/tutorial/)) および Python for Data Analysis:Wrangling with Pandas, Numpy and IPython (http://shop.oreilly.com/product/0636920050896.do)を参考に作成した。
このチュートリアルでは他のプログラミング言語を習得している方を想定し、Python の特徴などを説明する。

関数

関数(Function)は Python （に限らず多くのプログラム）においてプログラムの構造化、再利用に最も重要な方法である。 ざっくりいえば、似たプログラムコードを 2 回以上書くのであれば、関数でプログラムを書く価値がある。

関数は、`def 関数名(引数1, 引数2,...):`（ヘッダ行）ではじまり、その後コードブロックが続き、
戻り値は `return` で返す。
以下は、与えられた２つの引数から大きいものを返す関数:

In [1]:
```
def max(x, y):
    if(x > y ):
        return x
    else:
        return y

print(max(9,10))
```

```
10
```

Python では関数のなかにいくつ `return` があってもよい。 また、`return` なしにコードブロックの最後に到達すれば、`None` を返す。

In [1]:
```
def x_max(x, y):
    if(x > y ):
        return x

print(x_max(9,10))
```

```
None
```


Python の引数の与え方（再掲）

Python では引数は以下の3 種類の与え方が可能である。

1. 順番
   `f(10, 100, "foo", "bar")`
2. キーワード
   `f(keyword1 = 100, keyword3 = "foo", keyword2 = 100, keyword4 = "bar")`
3. 1.,2, の混在
   `f(10, 100, keyword4 = "bar",keyword3 = "foo")`

付け加えると、関数・オブジェクトメソッド側で引数の既定値が設定されていれば、呼び出し時にその引数を省略できる。
キーワードと、省略可能な引数は Pytgon の特徴の一つで、オプションの多い描画関係の関数・メソッドでしばしば登場する。

Python の引数定義

キーワード引数のキーワードは関数定義時の変数名が使われる:

In [3]:
```
def x_max(x, y):
    if(x > y ):
        return x
    else:
        return y

print(x_max(y = 9,x = 10))
```

```
10
```

また、既定値の設定は関数定義のヘッダ行でおこなう。
以下は `y` の既定値を `0` とした例:

In [6]:
```
def x_max(x, y = 0):
    if(x > y ):
        return x
    else:
        return y

print(x_max(-3))
```

```
0
```

変数のスコープ

関数は２つの異なるスコープ、global および local、の変数にアクセスする。 既定のスコープは local であり、以下の関数内のリスト `list_a` は関数を実行後破壊される。 組み込み関数 `id()` はオブジェクトの識別値を返す。標準的な実装では格納されているメモリアドレス:

In [9]:
```
list_a = None
def local_list():
    list_a = []             # 既定の local scope
    for i in range(5):
        list_a.append(i)
    print("inner function:", list_a, id(list_a))

local_list()
print("after function:",list_a, id(list_a))
```

```
inner function: [0, 1, 2, 3, 4] 4451763016
after function: None 4414560664
```

global 変数

関数から関数外の変数にアクセスするには、global 変数を宣言すればよい:

In [5]:
```
list_a = None
def local_list():
    global list_a           # global として宣言
    list_a = []
    for i in range(5):
        list_a.append(i)
    print("inner function:", list_a)

local_list()
print("after function:",list_a)
```

```
inner function: [0, 1, 2, 3, 4]
after function: [0, 1, 2, 3, 4]
```

複数の戻り値

関数で複数の戻り値が必要な場合、return に渡す結果を列挙すれば戻り値は tuple となる:

In [11]:
```
def multiple_returns():
    return 1, 2, 3

print(multiple_returns())
```

```
(1, 2, 3)
```

他の方法として、戻り値を辞書とすることもできる:

In [22]:
```
def multiple_returns():
    return {"apple":1, "orange":2, "peach":3}

print(multiple_returns())
```

```
{'apple': 1, 'orange': 2, 'peach': 3}
```


オブジェクトとしての関数

Python は関数もオブジェクトの一種である。関数をリスト・辞書のメンバー・値とすることもできる。
キーワードで呼び出し先の関数を変えたい場合は:

In [43]:
```
def print_apple():
    print("APPLE")

def print_orange():
    print("ORANGE")

def print_default():
    print("NONE")

dict_functions = {"apple":print_apple, "orange":print_orange}

for fruite in ["apple", "orange", "kiwi"]:
    dict_functions.get(fruite, print_default)()
```

```
APPLE
ORANGE
NONE
```

無名(Lambda)関数

Python では無名関数(Lambda 式)が利用できる。Lambda 式の書式は以下となる:

```
lambda 引数1, 引数2, .... : 引数を含む式
```

そして、以下の Lambda 式と関数定義は等価である:

```
f1 = lambda x, y : x + y

def f1(x, y):
    return x + y
```

Lambda 式はデータの並べ替えなどのソートキーの指定に利用されている。 以下のプログラムは、２重リストをソートするプログラムで、そーとキーは２つ目の要素となっている:

In [7]:
```
list_input = [["apple", 100], ["orange", 50], ["kiwi", 200]]

print(sorted(list_input, key=lambda x:x[1]))
```

```
[['orange', 50], ['apple', 100], ['kiwi', 200]]
```


Generator

Python の重要な特徴としてリストやファイルの行にまたがるような繰り返し処理の一貫した取り扱いが挙げられる。 generator によって繰り返しに対応した関数を簡単に構成できる。 通常の関数は `return` によって結果を返し、動作を終了するが、 generator は複数の戻り値を順番に返す、そして次の戻り値の要求があるまで動作は停止する。 generator を生成するには、`return` に代えて `yield` を使えば良い：

In [11]:
```
def squares(n):
    for i in range (0, n):
        yield i**2

for s in squares(10):
    print(s)
```

```
0
1
4
9
16
25
36
49
64
81
```

generator 式

Python ではタプル、`()`、で内包表記を利用することはできないが、 小括弧内`()`に内包表記（厳密には違う）を記述するとタプルではなく generator オブジェクトがつくられる。

したがって、以下のプログラムは動作するがオブジェクト `squares_gen` はタプルではない。

```
Python:generator_sample.py
squares_gen = (x**2 for x in range(6))
for x in squares_gen:
    print(x)
```

エラーおよび例外処理

たとえば、以下の型変換処理はエラーとなり、プログラムは終了する:

In [12]:
```
int("abc")
```

```
---------------------------------------------------------------------
----------
ValueError                                Traceback (most recent
call last)
<ipython-input-12-2dda1cc00c48> in <module>()
----> 1 int("abc")

ValueError: invalid literal for int() with base 10: 'abc'
```

Python でエラー発生時の例外処理は、`try`, `except` で記述できる。
以下は型変換でエラーが生じた場合は、引数をそのまま返す関数 `example_f()`の例:

In [13]:
```
def example_f(x):
    try:
        return int(x)
    except:
        return(x)

example_f("abc")
```

Out[13]:
```
'abc'
```

エラーごとに処理を変えたい場合は`except` にエラーコードを加えて並記していく:

In [14]:
```
def example_f2(x):
    try:
        return int(x)
    except (ValueError, TypeError):
        return(x)
    except SystemError:
        return(None)

example_f2("abc")
```

Out[14]:
```
'abc'
```

エラーおよび例外生成

Python で簡単に例外を発生させるには、`assert` 条件式 を利用する。 条件が `False` のときに例外が発生する。

In [23]:
```
def just_assert( x, y):
    assert x == y

print(just_assert(1,1))

print(just_assert(1,0))
```

```
None
---------------------------------------------------------------------
----------
AssertionError                            Traceback (most recent
call last)
<ipython-input-23-58fd6cb1a181> in <module>()
      4 print(just_assert(1,1))
      5
----> 6 print(just_assert(1,0))

<ipython-input-23-58fd6cb1a181> in just_assert(x, y)
      1 def just_assert( x, y):
----> 2     assert x == y
      3
      4 print(just_assert(1,1))
      5

AssertionError:
```

