第5章組み込み型を使いこなす

12T4069L
佐鳥　恭太郎

数値の種類

1023	10進数
0b11111111	2進数
0o1777	8進数
0x3ff	16進数

どれも1023

10進数変換

int(“数値”, 進数) => 10進数

“数値”	進数
“0bXXXX”	2
“0oXXXX”	8
“0xXXXX”	16

0b,0o,0xは省略可
数値の文字列と進数が合ってればよい

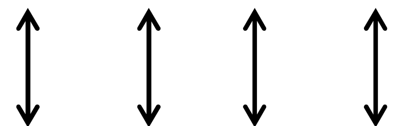
各進数変換

bin(数値) => ‘2進数文字列’
oct(数値) => ‘8進数文字列’
hex(数値) => ’16進数文字列’

ビット演算

x|y	xとyの論理和(OR)を取る
x&y	xとyの論理積(AND)を取る
x^y	xとyの排他的論理和(XOR)を取る
x<<y, x>>y	xをy分だけ左シフト(<<<),右シフト(>>>)
~x	-(x+1) を返す。*ビット反転ではない

x = 0 b X X X X



各ビットの演算

y = 0 b X X X X

整数型 z = 0 b X X X X

ビット演算の諸注意

桁数が合わないときは0が補完される

		x	=	0	b	1	0	1	0	1
		y	=	0	b		0	0	1	1
x	&	y	=	0	b	0	0	0	0	1

ビット反転~

Pythonの整数の桁は無制限なので反転ができない

=>~x = -(x+1)　と定められている

例:~0b1000(8) = -0b1001(-9)

=>ビット反転したいときは

x ^ 0b1111...(xのビット数と同じ桁数)

=>x ^ (1 << x.bit_length()) – 1

とする。

xのビット桁数

ビット演算は
算術演算より
優先度が低い

Pythonでの整数

Python2.6, 2.7系での整数型はintとlong型がある。

>>>import sys

>>>sys.maxint

[int型の最大値]

sys.maxintを超えると変数がlong型になる

long型の最大値はメモリの限界まで

Python3系ではlong型が廃止され、int型のみ（最大値なし）

エスケープシーケンス

```
print ('Hello
World')
=>ERROR
```

```
print ("'Hello
World"')
=>
'Hello
World'
```

改行も出力された

インデントしていないから見づらい
⇔インデントするとそれも空白（Tab）として認識されて出力される

```
print ("'Hello
		World"')
=>
'Hello
		World'
```

エスケープシーケンスの種類

エスケープシーケンス	説明
¥n	改行
¥r	改行（CR,キャリッジリターン）
¥t	水平タブ
¥f	改ページ(フォーム・フィード)
¥' , ¥"	シングル(ダブル)クオーテーション
¥¥	バックスラッシュ
¥x61	8バイトの16進数に対応する8ビット文字
¥u3042	16ビット16進数に対応するUnicode文字、16進数部分には「0x」は不要
¥0	Null文字

raw文字列

’¥記号’が多くあると見づらい or 記号をそのまま出力したい …

r’C:¥path¥to¥file’
=>’C:¥path¥to¥file’

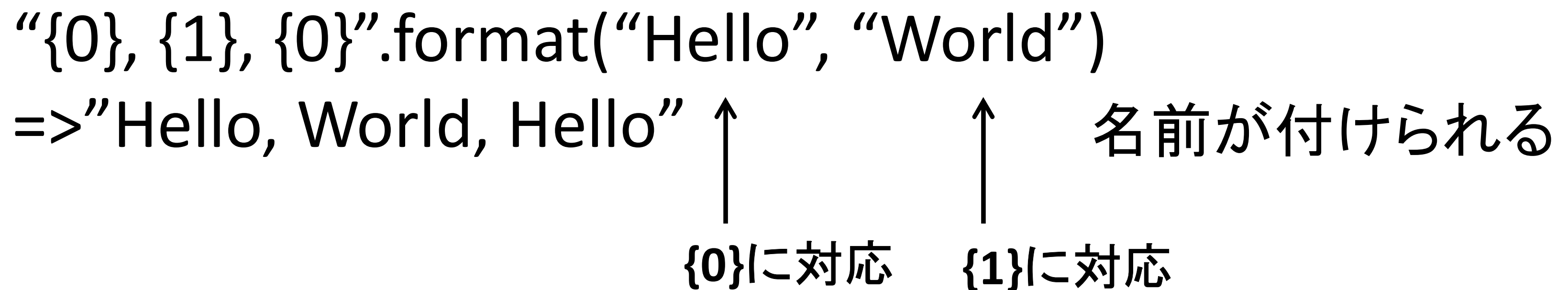
rの後ろの文字列の中身をそのまま出力する

文字列のメソッド

メソッド	説明
str.find()	strの先頭から先頭から検索を行う。見つかったらインデックスを返す。 なかったら-1を返す。
str.index()	str.find()と大体同じ。見つからない場合に例外を発生
str.endwith()	strの最後尾が検索対象ならTrue,でなかったらFalse
str.startwith()	strの先頭が検索対象ならTrue,でなかったらFalse
str.split() str.rsplit()	strを指定した区切り文字で区切った文字列のリストを返す。分割数を指定できる。指定しない場合は最後まで分割する。rsplit()は末尾から。
str.join()	与えられたシーケンスをsで連結する。連結された文字列が返される。
str.strip()	strの先頭及び末尾から指定した文字列を削除する。
str.upper()	strの英字小文字を大文字に変換した文字列を返す
str.lower()	strの英字大文字を小文字に変換した文字列を返す
str.ljust() str.rjust() str.center()	strを指定した幅を考慮して左寄せした文字列を返す 同様に右寄せした文字列を返す。 同様に中央寄せした文字列を返す

format

=>”Hello, World”



formatにシーケンス

d = {“name”:”Guido”, “birthyear”:1964}

“{0[birthyear} is {0[name]}’s birthyear.”.format(d)

辞書の中身をこっちでも見れる

辞書を渡してあげると

=>”1964 is Guido’s birthyear.”

辞書と同様にオブジェクトの持つ属性を指定することができる。

import sys
“Python version: {0.version}”.format(sys)

formatの表記法

“{0:03}”.format(1)

=>”001”

コロンを付けてあげると表記法を指定できる。

表記法の種類

オプション	説明
<	スペース(デフォルト)で左詰め　{:[補う文字]<[数]}
>	スペースで右詰め　{:[補う文字]<[数]}
^	中央寄せ　{:[補う文字]<[数]}
+	数値に符号を付ける
-	数値が負のときだけ符号を付ける
空白	数値が正のときは空白、負のときは符号を付ける
c	要素を文字列として埋め込む
d	要素を10進整数として埋め込む。入力が小数点を含む、文字列だとエラー。
f	要素を10進整数として埋め込む。小数点を含む数値を扱える。
x	要素を16進数文字列として埋め込む。xで英字が小文字に、Xで大文字。
b	要素を2進数文字列として埋め込む。
%	要素をパーセンテージとして埋め込む。
,	数値の千の位にカンマを付けて埋め込む。

リスト,タプル操作

スライス

list = [0, 1, 2, 3, 4, 5]

list[0:5:2]

インデックスが2個進むごとに取り出す

インデックス5未満

インデックス0以上

=>[0, 2, 4]

リストのメソッド

メソッド	説明
list.sort(key=関数)	keyには引数が１つだけの関数を指定できる。第２引数にTrueを渡すと結果を逆順にできる。
list.reverse()	リストの並び順を逆順に書き換える
list.remove()	リストから引数に与えたものを削除する。見つからない場合は例外(エラー)が発生。
list.append()	リストに引数に与えたものを末尾に追加する。
list.extend()	list.append()とは違い、引数にシーケンスを与え、複数の要素を末尾に追加する。
list.pop()	リストから引数で与えられたインデックスの要素を取り除いて、その要素を返す。引数を指定しないと末尾から取り出す。
list.index()	リストの要素を検索し、インデックスを返す。見つからないときは例外(ValueError)を発生。

set型

集合型

set = {1, 2, 3, 4}

同じ要素はない

順番がない=>インデックスで指定できない

set型同士で&,|,-の演算ができる

セット（集合）のメソッド

メソッド	説明
set.union(set2)	setと引数の集合の和集合を返す。 set | set2　と同値。
set.intersection(set2)	setと引数の集合の共通集合を返す。 set & set2　と同値。
set.difference(set2)	setと引数の集合と差集合を返す。 set – set2　と同値。
set.symmetric_difference(set2)	setと引数の集合の対称的差集合（どちらか一方に含まれる要素の集合）を返す。 set ^ set2と同値。
set.add()	setに引数の要素を追加する。
set.remove()	setから引数の要素を取り除く。要素がなかった場合は例外（KeyError）が発生。set.discard()を使うと例外は発生しない。

辞書

d = {key:val}

d = dict(辞書オブジェクト) # コピー

d = dict((key,val), (key,val), ...)

d = dict(key=val, key=val)

＝を使用しているので数値をkeyに設定できない
このkeyだけはは文字列ではなく
dict(one=1, two=2)の様にキーにしたい文字を書く
valは自由

辞書のメソッド

メソッド	説明
dict.keys()	辞書のキーの一覧を返す。
dict.get(キー[,値])	辞書からキーに対応した値を取り出す。なかった場合第2引数の値を返す。それもない場合Noneを返す。
dict.setdefault(キー[,値])	辞書からキーに対応した値を取り出す。 なかった場合第2引数の値を返す。加えて、キーを新規登録して値を代入する。
dict.items()	辞書に登録されているキーと値をペアにしたタプルをリストにして返す。
dict.values()	辞書の値の一覧を返す。
dict.update([要素])	辞書を要素で上書きする。dictに存在するキーは上書き、存在しないキーは新規登録される。

辞書メソッドの活用

```
for word in line.split() :
        if word in wordcount :
                wordcount[word] = wordcount[word] + 1
        else :
                wordcount[word] = 1
```

```
for word in line.split() :
        wordcount[word] = wordcount.get(word, 0) + 1
```

TrueとFalse

Pythonは文字列が空のとき、Falseとみなす。
=>if len(s) != 0 :
⇔if s :

True
- 0以外の数値
- 空でない文字列
- 要素を持つシーケンス（リストやタプル）,辞書

False
- Trueの逆

range()

range()を使ってうまくループを回す

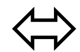
range([初期値,] 終了する数値 [, ステップ])

Xからyまでステップずつ増える（減る）カウンタができる

シーケンスとループカウンタ

```
counter = 0
for item in seq :
        …
        counter += 1
```

```
for cnt in range(len(seq)) :
        print (seq[cnt])
```



```
for cnt, item in enumerate(seq) :
        print(cnt, item)
```

zip()

zip(シーケンサ1, シーケンサ2)

=>

[(シーケンサ1[0], シーケンサ2[0]),

(シーケンサ1[1], シーケンサ2[1]),

...]

2つのシーケンサの先頭から順番にタプルのリストで要素が取り出される。

どっちかのシーケンサの要素が尽きたら終了

関数の戻り値

戻り値はリストで返せる

```
def foo() :
        a, b = 1, 2
        return [a, b] # return a, b と書くとタプルで返る
alist = foo()
=>alist = [1,2]
```

アンパック代入

```
c, d = foo()
```

関数の引数

```
def foo(a, b, *vals) : # 引数にアスタリスクを付ける
        print(a, b, vals)    一番後ろでないとERROR

foo(1,2,3,4,5)
=>1 2 (3, 4, 5)
```

valsにタプルとして入力される

```
foo(1,2,c=3)
=>ERROR
```

未定義のキーワードとして入力できない

未定義なキーワード引数

```
def foo(a, b, **args) : #引数にアスタリスクを2つ
        print(a, b, args)

foo(1, 2, c=3, d=4) #キーワード指定が可能に
=>1 2 {‘c’:3, ‘d’:4}
```

Pythonの日本語

数字,アルファベット,よく使う記号,制御文字

=>asciiコード, 1バイト文字（8bit:MSBは0)

日本語

=>shift-jis, utf-8 :2バイト文字

文字列型とバイト型

ユニコード

=> 文字列型, １文字の長さ1

shift-jis, utf-8, euc-jpなど

=> バイト型, １バイトの長さ1

=>2バイト文字の長さが変わってくる

len(ユニコードの日本語１文字) => 1

len(バイト型の日本語１文字) => 2

日本語対応エンコードの種類

エンコード名	説明
シフトJIS	2バイト文字。第1バイトが特定の範囲ならば第2バイトと合わせて表現する。それいがいなら1バイトでasciiコードと同じ。
EUC-JP	UNIX,LINUXで利用されていたもの。文字列中に指定のコードをいれて文字の種類を切り替えている。asciiと共存
UTF-8	Unicodeをベースにしたエンコード。様々な言語に対応しており、必用ならば3バイト使って文字を表すこともある。

ユニコード

ユニコードは多くの言語を表せ、他のエンコードと簡単に相互変換できるため優秀

ユニコード文字列の作り方

s = unicode(‘文字列’) = u’文字列’

ユニコード文字列 => エンコード文字列

s.encode(‘エンコード名’)

エンコード名の例:shift-jis, utf-8, euc-jp,など

‘エンコード名’で書かれた文字列 => unicode文字列

‘文字列’.decode(‘エンコード名’)

例外処理

str.encode([‘エンコード名’[, ‘エラー処理の方法’]])

“文字列”.decode([‘エンコード名’[, ’エラー処理の方法’]])

=>第二引数で、エンコード中に例外が発生した時の処理法が指定できる

エラー処理の方法

文字列	説明
strict	デフォルト設定。エラーが発生したら変換を停止する
replace	変換できない文字は適切な文字列(「?」など)に置き換えて変換を続行する
ignore	変換できない文字は取り除かれ変換を続行する

エンコーディング指定

スクリプトファイルのエンコーディングを指定する
ファイルの一行目または二行目に以下のコメントを書く

```
# coding: エンコード名
# coding=エンコード
```

ここの前後は自由なので

```
# -*- coding: エンコード名 -*-
```

のようにも書ける

デフォルトの文字コード

Pythonでのデフォルトの文字コードは

```
>>>import sys
>>>sys.getdefaultencoding()
'ascii'
```

で見れる。

プログラム内で変更するには

```
>>>import sys
>>>reload(sys)
>>>sys.setdefaultencoding('文字コード')
```

とする。

=>encode(),decode()メソッドでのデフォルトの文字コード指定がここで設定した文字コードになる

文字列符号化

str型の文字列符号化はプラットフォームによって異なる。

Mac系=>utf-8

Windows=>cp932

バージョンによる文字の扱い

Python2系

- 文字列(str)型(入力された文字コードを保持)
- unicode型(常にunicode型で保持)

Python3系

- 文字列型(内部的にunicodeで統一)
- bytes型(内部的にバイト列として扱う)

=>内部でunicodeに統一されたので処理が簡単になった。