Table of Contents

Table of Contents

AtlasJapanSoftwareTutorial16DecPython

C++ も簡単 け コ ルせず &#x5B9F

- コ ルせず 実行可能
- 豊富なラ ブラリ
- 文法 C++ も単純

と った便利な点があ が &#x5B
実行速度が必要な大規模&#
(ミュ レ や 再構成 コア&#x90
C++ 関数 呼び出 使う とが&#x3
た計算機で走らせる各種

最新バ 3.5で が ATLASで 2.7 標準

た 他 リプ 言語と比較 る&

- オブ ェ 指向
- き方 柔軟性が少な ため
  (例えば PERL 逆)

と った特徴があ

実行方法

通常 プ グラム ファ ル 実&

```
$ setupATLAS
$ localSetupPython
$ python
>>> print "hello"
hello
>>> CTRL-d # # to exit python
```

で 同じ と リプ ら実行 ょ&

```
$ python hello.py
```

一番最初 行 リプ コマ ド

- `hello.py` 中身:

```
#!/usr/bin/env python
# This script prints hello to the screen
print "hello"
# EOF #
```

- コマ ド 実行:

```
$ chmod +x hello.py
$ ./hello.py
hello
```

文法 基本

- ハッ ュ(#
) も後ろ 部分 行 終わ で&#x3
- 改行がコマ ド文 終わ &#x306A
  (C++で 改行 意味 な)
- 1行 複数 文 きた 場合 セ&#x30
  ;)で分ける
- 一つ 文 複数行 続けた 場&
- ブ ッ (ifやfor 中身)
  同じ デ で決めら る
  (C++ う { } で囲むわけで な)
  - ♦ タブで なく 4つが推奨
- っ 使った場合
  っ が閉じら る でブ ッ &#x3
  () [] {}

```
l = [ "muon",
      "electron" ]
```

基本的な変&#x6570

- **int**: 整数型(C++ long 対応) も大きな整数 long が使わ る
- **float**: 64ビッ 浮動小数点型 (C++ double)
- **complex**: 複素数
- **str**: ミュ タブル 文字列, C++ const char* みた なも
- **bool**: ブ リア True False
- **list**: (通常 同じ型) デ タ 列 (C++ std::vector みた なも)
- **tuple**: ミュ タブルなデ タ列 (C++ 構造体みた な使 方)
- **dict**: キ と値 ア 列, (C++ std::map<K,V> みた なも)
- ある変数 型 **type** 関数で知る とができる

```
>>> type("hello")
<type 'str'>
>>> type("hello").__name__
'str'
```

ミュ タブルと &#x3

ミュ タブル(値 変更可能) float, complex, str, bool, tuple等 多く 型 変数が ミュ タ&#x 例えば

```
>>> v = 10
>>> v = 20
```

と た場合 v 値が変わっ る &#123 見方 C++等多く 言語と違う &#12

C++や多く 他 リプ 言語で 変 例えば

```
>>> v = 10
>>> type(v)
>>> v = v + 2.2
>>> type(v)
```

と た場合 一行目 実行後 v &#x5

演算子

- **+** , **-** , ***** , **/** , ** : 四則演算, 累乗
- `+=` , `-=` , `*=` , `/=` : 演算と代入
  (=C++=と同様)
- **%**: (int 対)割 算 余 ,
  (文字列 対)フォ マッ 指&#x5E
- **or and**: 論理和と積
- **not**: 否定
- **<** `<=` **>** `>=` : 大小比較
- **==** , `!=` : 同値 非同値
- **is**, **is not**:
  同じオブ ェ 指 る どう
- **in**, **not in**:
  ある要素が集合 含 る &#x306
- **| ^ & ~ << >>**: bit演算子
- **X < Y < Z**: `Y` が `X` and `Z` 間 ある場合 真

```
>>> u = 10
>>> v = 10.0
>>> u == v
>>> u is v
```

リ

リ C++ **std::vector<T*>** 似 値 列 格納 そ 値も変更異なる型も格納 る とがで&

```
>>> v = []       # empty list
>>> v = list()   # empty list
>>> v = [1, 2, 3, 4, 5]
>>> v = ['a', 'b', 'c']
>>> v = range(4, 10, 2)    # results in [4, 6, 8]
>>> print v
>>> v = [4, 2.5, "Hi", [1,3,5]] # can mix types
>>> print v
```

要素 追加

```
>>> v = range(4)
>>> print v
[0, 1, 2, 3]
>>> v.append(70)
>>> print v
[0, 1, 2, 3, 70]
```

リ 結合

```
>>> v = range(4)
>>> print v
[0, 1, 2, 3]
>>> v += [5, 6, 7]
>>> print v
[0, 1, 2, 3, 5, 6, 7]
```

要素 削除

```
>>> v = [4, 2.5, "hi"]
>>> v.remove(2.5)
>>> print v
[4, 'hi']
>>> del v[0]
>>> print v
['hi']
```

サ ズ 取得

```
>>> v = [4, 2.5, "hi"]
>>> len(v)
3
```

要素 読み き

```
>>> v = range(4,8)
>>> print v
[4, 5, 6, 7]
>>> v[0]
4
>>> v[0] = "hey"
>>> print v
```

```
['hey', 5, 6, 7]
>>> v[-1]           # last element. negative index: count from the end
7
>>> v[1:3]          # subrange by index (start idx, one-beyond-last idx)
[5, 6]
>>> v[1:3] = ["a", "b"]
>>> print v
['hey', 'a', 'b', 7]
```

要素 存在 テ

```
>>> v = range(4,8)
>>> if 4 in v:
...  print "found it"
...
found it

>>> if 200 not in v:
...  print "not found"
...
not found
```

要素 対 るル プ&#x

```
>>> v = range(4)
>>> for i in v:
...  print i
...
0
1
2
3
```

で 扱 せんが リ 対 C++ vectorと&#x5
(**sort** , **reverse** , **index** , **count** , ...).
以下 コマ ドで詳 説明 見&#x

```
>>> help(list)
```

リ　作成　る内&#

で　内包表現　使う　とでリ　&#

```
>>> v = [x**2 for x in range(10) if x % 3 == 0]
>>> print v
```

と　ると　10　で　3　倍数　二乗　リ　&#x304

文字列

- 文字列 グル ォ (') ブル &#x30A9 ''' """):

```
>>> a = 'hello'
>>> b = "how are you?"
>>> c = '''first line
... second line'''
>>> d = """look ma! another
... line"""
>>> print a
hello
>>>print b
how are you?
>>> print c
first line
second line
>>> print d
look ma! another
line

>>> aa = "'"; bb = '"'
>>>print aa
'
>>> print bb
"

>>> aa = "\""; bb = '\''
```

結合

```
>>> a = 'hello'
>>> d = a + "! "
>>> d                  # will show object on screen
'hello! '              # note the quotes
>>> b = 'how are you?'
>>> d += b
>>> print d
hello! how are you?  # no quotes printed here
```

要素へ ア セ

```
>>> d[1]              # read-only. d[0] is first character
'e'
>>> d[1] = '2'        # will fail
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

部分文字列

```
>>> d
'hello! how are you?'

>>> d[7:10]  # start at char at index 7 (8th char), up to
             # (but not including) char at index 10 (11th char)
'how'

>>> d[-4:]   # start at 4th char from the end, continues until the end
'you?'
```

検索

```
>>> d
'hello! how are you?'

>>> d.find('o')     # returns the index of the first match
4
>>> d.find('H')     # returns -1 if not found (case sensitive)
-1
>>> d.find('how')   # returns index of first character
7
```

C言語風 フォ マ&#

```
>>> print "%6.4f %4d" % (5.2, 200)
5.2000  200
```

後で た説明がある が

- `"%s"` そ ラ __str__が返 文字列
- `"%r"` (calls __repr__が返 文字列が入

も 特 フォ マッ が必要な &#2258

```
>>> number = 5.6
>>> str(number)
```

う る とで数字 文字列 変&#255

繰 返

```
>>> 10 * "#"
'##########'
```

文字列 分割

```
>>> d
'hello! how are you?'

>>> d.split()           # default split character is whitespace
['hello!', 'how', 'are', 'you?']

>>> d.split('h')
['', 'ello! ', 'ow are you?']
```

文字列 連結

```
>>> ds = d.split()
>>> ds
['hello!', 'how', 'are', 'you?']
>>> "---".join(ds)
'hello!---how---are---you?'
```

辞 型

辞 型
キ と値 ア 集合で 例えば &#
dict 作

```
>>> k = {}            # empty dictionary
>>> k = dict()        # empty dict
>>> k
{}
>>> k = {'muon': 4, 'electron': 2}  # key: value
>>> k
{'electron': 2, 'muon': 4}
>>> k['muon']
4
```

要素 操作

```
>>> k['jet'] = 1          # add or overwite an entry
>>> k
{'electron': 2, 'jet': 1, 'muon': 4}

>>> k.update({'tau':1, 'photon': 2}) # add from another dict
>>> k
{'electron': 2, 'jet': 1, 'muon': 4, 'photon': 2, 'tau': 1}

>>> k.setdefault('muon', 1)          # only sets if not already there
4
>>> k
{'electron': 2, 'jet': 1, 'muon': 4, 'photon': 2, 'tau': 1}

>>> k.setdefault('higgs', 42)        # only sets if not already there
42
>>> k
{'electron': 2, 'higgs': 42, 'jet': 1, 'muon': 4, 'photon': 2, 'tau': 1}
```

dict へ ア セ

```
>>> k.get('muon', 0)         # returns 0 if 'muon' is not in dict
4

>>> k.keys()                 # returns the keys in a list
['muon', 'jet', 'higgs', 'photon', 'tau', 'electron']

>>> k.values()               # returns the values in a list
[4, 1, 42, 2, 1, 2]

>>> k.items()                # returns the items: a list of (key, value) tuples
[('muon', 4),
 ('jet', 1),
 ('higgs', 42),
 ('photon', 2),
 ('tau', 1),
 ('electron', 2)]
```

要素 削除

```
>>> k
{'muon': 4, 'jet': 1, 'higgs': 42, 'photon': 2, 'tau': 1, 'electron': 2}

>>> del k['electron']
```

```
>>> k
{'muon': 4, 'jet': 1, 'higgs': 42, 'photon': 2, 'tau': 1}
```

繰 返 操作

```
>>> k
{'higgs': 42, 'jet': 1, 'muon': 4, 'photon': 2, 'tau': 1}

>>> for n in k:         # loop over keys
...  print n, k[n]      # ok, bad example: use items() to access both the key and the value!!
...
muon 4
jet 1
higgs 42
photon 2
tau 1

>>> for n,v in k.items():  # n,v is an implicit tuple filled for every pass in the loop
...  print "k[%r]= %r" % (n,v)
...
k['muon']= 4
k['jet']= 1
k['higgs']= 42
k['photon']= 2
k['tau']= 1
```

dict 要素 検査

```
>>> k
{'higgs': 42, 'jet': 1, 'muon': 4, 'photon': 2, 'tau': 1}

>>> if 'muon' in k:           # searches in the keys
...  print "got a muon"
...
got a muon

>>> if 'electron' not in k:
...  print "no electron"
... else:
...  print "got an electron"
...
no electron
```

ブ リア

ブ ル値 **True** と **False** と く (C++ と違 最初 大文字)

そ ぞ 型 **True** と **False** 対応 た値 持っ る:

Type	**True**	**False**
int, **long**, **float**	non-zero	0
list	non-empty	empty **[]**
tuple	non-empty	empty **()**
dict	non-empty	empty **{}**
str	non-empty	empty **""**

- **X** or **Y**:
  - **X** and **Y** がどんな型でも
  - 返 値 **X** が **True** 時 **X** (そ 場合 **Y** 評価さ な), そうでな 場合 **Y** が返さ る
  - 返 値 型 **X** **Y** 型 必ず も **bool** で な !!
- **X** and **Y**:
  - **or** とほぼ同じだが :
  - 返 値 **X** が **False** 時 **X** で **Y** 評価さ ず 返 値 **Y**

変数

- 変数 型 明示的 指定 る と
- 変数 型 動的 変化 る
  - 変数 型 人が指定 る と
  - 変数 型 そ 時 参照 る&#x30A
  - 変数 型 変更可能

```
>>> a = "hello"
>>> type(a)
<type 'str'>
>>> a = 5
>>> type(a)
<type 'int'>

>>> b = a      # b = 5, a and b now point to the same object
>>> a = 7      # does this change the object pointed to ?
>>> print b    # what should this print ? 5 ? or 7 ?
```

Show solution Hide

```
>>> print b    # yep, 5. a has been 'rebound' to a new object,
               # but b has been left untouched, so it still points to
          # the object 'the integer 5'
5
```

- `list` と `dict`
  ミュ タブルな型である
  (そ でも変数 実際 オブ &#x30A7

```
>>> a = [1, 2, 3, 4]
>>> b = a               # b now points to the same list
>>> b[3] = 6            # changes an element in the list
>>> print a             # a is changed
[1, 2, 3, 6]
```

- - 場合 `b`
    変更 加えた時 `a`
    も変化 る
  - 他 操作 時も同様 例え&#
    `b.append(7)` や **`b += [8,9]`**

- で 文字列 場合 どうで ょ&#

```
>>> a = "hello"
>>> b = a
>>> a += " !"
>>> b   # what should this print ?
```

Show solution Hide

```
>>> b   # yep, 'hello'. a has be 'rebound' to a new object (as strings
        # are immutable), but b has been left untouched, so it still points
   # to the object 'the original string "hello"'
'hello'
```

コピ　作成

で　今行った　うな実際　オ&#x30

リ

```
>>> v1 = [1, 2, 3, 4]
>>> v2 = v1[:]        # makes a copy ('sub'-list of the complete original list)
>>> v3 = list(v1)     # another way to make a copy
>>> v2[3] = 8         # only modifies v2, leaves v1 and v3 unchanged
>>> v3[3] = 12        # only modifies v3, leaves v1 and v2 unchanged
>>> v1
[1, 2, 3, 4]
>>> v2
[1, 2, 3, 8]
>>> v3
[1, 2, 3, 12]

>>> import copy
>>> v4 = copy.copy(v1)
>>> v1[3] = 0
>>> v4
[1, 2, 3, 4]
```

辞　型

```
>>> d1 = {"muon": 2, "electron": 2}
>>> d2 = d1.copy()
>>> d3 = dict(d1)
>>> d4 = copy.copy(d1)
>>> d1["muon"] = 22

>>> d1
{'electron': 2, 'muon': 22}
>>> d2
{'electron': 2, 'muon': 2}
>>> d3
{'electron': 2, 'muon': 2}
>>> d4
{'electron': 2, 'muon': 2}
```

浅　コピ　と深　コ&#x

- 今　で見　きたコピ　浅　コ&#x30
  `list` や `dict`
  作　元　要素へ　参照　き込&#x30
- **`v5 = copy.deepcopy(v1)`**
  うな深　コピ　ると要素&#x30

```
>>> d = {"cutflow": [10, 5, 3]}
>>> d1 = dict(d)
>>> d2 = copy.deepcopy(d)

>>> d["cutflow"][2] = 0
>>> d, d1, d2 # what should they print ?
```

Show solution Hide

```
>>> d
{'cutflow': [10, 5, 0]}
>>> d1
```

```
{'cutflow': [10, 5, 0]}
>>> d2
{'cutflow': [10, 5, 3]}
```

メモリ管理

python で (C++と違っ)
ガ ッ コレ タ っ メモリが&
実際 python
がオブ ェ へ 参照 数 数え

```
>>> a = [1,2]
>>> b = a
>>> del a   # deletes the variable a, not the object
>>> del b   # deletes last reference, hence deletes the object too
```

分岐とル プ

if-then-else

```
>>> d1 = {"muon": 2, "electron": 3}
>>> if "muon" in d1:
...  print "found a muon"
... elif "electron" in d1:
...  print "found an electron but no muon"
... else:
...  print "no muon and no electron"
...                                  # what should this print ?
```

Show solution Hide

```
found a muon
```

forル プ

ら先 コ ド 見るだけで進&#1241

```
for k,v in d1.items():     # loop over items in d1
    if k == "muon":
        process_muon(v)
        continue           # go to next item in the dict
    if k == "unknown":
        break              # bail out of the loop
else:
    # executed if no 'break' was executed
    print "All OK: no 'unknown' particles"
```

関数

関数 定義

```
def muonSelector(particle, ptmin=50*GeV):
    """
    muonSelector selects particles which are muons with a
    pt greater or equal to `ptmin`
    """
    if particle.type() == "muon":
       if particle.pt() &gt;= ptmin:
          return True
    return False
```

- **def**
  キ ワ ド 後 部分 新 関数 &#x
- 最初 新 関数 名前と引数
- 関数 説明 先頭 おくと&#x30
- そ 次 関数 中身が る

- さっき **muonSelector** 関数で **particle**
  呼び出 際 必ず引数と &#x6E21
- **ptmin**
  必須で なく 省略さ た場&
  50*GeV が使わ る

関数 可変長 引数やキ ワ &#x30C

```
>>> def function(a, b, c):
...     print a
...     print b
...     print c
```

以上 うな関数 以下 う 呼&#1240

```
>>> function(c=1, a=2, b=3)
2
3
1
```

う キ ワ ド指定 ると 順序&#12

```
def complicated_fct(arg1, *args, **kwargs):
    """my documentation"""
    if arg1 &gt; 10 and len(args) &gt; 0:
        return args[0] == 42
    if 'ptmin' in kwargs:
        return kwargs['ptmin'] &gt; 50.*GeV
    return

# example: args[0]==55., kwargs['ptmin']==25.*GeV
complicated(5, 55., ptmin=25.*GeV)

# example: len(args)==0, kwargs['ptmin']==25.*GeV
complicated(5, ptmin=25.*GeV)
```

- 関数 定義で 引数 型 指&#x5B9A
- 関数 返 値も同様 型 指定

- **return** 文がな 場合 **None** が返さ る

そ で 色々な関数 見 き ょ&#x

```
### a variadic function --------------------------------
>>> def one(*args):
...     print args

>>> one()
()
>>> one(0)
(0,)
>>> one(0, 1, 2)
(0, 1, 2)

### a function with arguments with default values ------
>>> def two(b=1, c=2):
...  print "b=%s c=%s" % (b, c)

>>> two()
b=1 c=2
>>> two(0,1)
b=0 c=1
>>> two(b=2, c=3)
b=2 c=3
>>> two(c=2, b=3)
b=3 c=2

### a variadic function, with keyword arguments ---------
>>> def three(*args, **kwds):
...     print "args=%s" % (args,)
...     print "kwds=%s" % (kwds,)

>>> three(0)
args=(0,)
kwds={}
>>> three(0,1)
args=(0, 1)
kwds={}
>>> three(0, 1, a=1, b=2)
args=(0, 1)
kwds={'a': 1, 'b': 2}
>>> three(a=1, b=2)
args=()
kwds={'a': 1, 'b': 2}

### everything in the mix -------------------------------
>>> def four(a, b=1, *args, **kwds):
...     print "a=%s b=%s" % (a, b)
...     print "args=%s" % (args,)
...     print "kwds=%s" % (kwds,)

>>> four(0)
a=0 b=1
args=()
kwds={}
>>> four(0, 2)
a=0 b=2
args=()
kwds={}
>>> four(0, 2, 3, 4)
a=0 b=2
args=(3, 4)
kwds={}
>>> four(0, 2, 3, 4, c=2, d=3)
```

```
a=0 b=2
args=(3, 4)
kwds={'c': 2, 'd': 3}
>>> four(0, c=2, d=2)
a=0 b=1
args=()
kwds={'c': 2, 'd': 2}
>>> four(a=2, b=42, zzz=666)
a=2 b=42
args=()
kwds={'zzz': 666}
>>> four(b=2, a=32)
a=32 b=2
args=()
kwds={}
```

可変長引数 持つ関数 便利&#

ミュ タブルな&#x5

ミュ タブルな変数が引数 &#
ミュ タブルな変数 場合 呼&

ミュ タブルな型(`list` `dict`
) デフォル 値と 持つ関数 &#x30

```
>>> def f(a, L=[]):
...     L.append(a)
...     return L
...
>>> print f(1)
[1]

>>> print f(2)      # what should this print ?
```

Show solution Hide

```
>>> print f(2)
[1, 2]

>>> print f(3)
[1, 2, 3]

# default values are evaluated ONCE.
# this means 'L' is always pointing at the same object
# in which we repeatedly append elements.

# prefer this way of passing mutable default values:
>>> def f(a, L=None):
...     if L is None: L = []
...     L.append(a)
...     return L
```

最後 例 う ると L 毎回空 &#x30EA

ラ

ラ 以下 う 定義でき

```
>>> class Track(object):
...     """Track represents a track with p, eta and phi components"""
...     def __init__(self, p=0, eta=0, phi=0):
...         """create a new track with p,eta,phi"""
...         self.p = p
...         self.eta = eta
...         self.phi = phi

>>> help(Track)
>>> help(Track.__init__)

>>> t1 = Track()
>>> t2 = Track(200)
>>> t3 = Track(200, 0.5, 1.2)
>>> print t1
&lt;__main__.Track instance at 0x1007ad440&gt;
>>> print t1.p
0
>>> print t1.phi
0
>>> print t3.phi
1.2
>>> print t3
&lt;__main__.Track instance at 0x1007ad128&gt;

>>> print t1.__init__
<bound method Track.__init__ of <__main__.Track instance at 0x1007ad440>&gt;
>>> print t1.__dict__
{'p': 0, 'phi': 0, 'eta': 0}
```

- __init__ ラ コ ラ タで
- コ ラ タ 最初 引数 (**self**, by convention)
  オブ ェ 自身 参照で必&#x30
- デ タメ バ __init__ 中で動的 &#x8FF
  が追加さ たデ タメ バ &#x683

```
>>> t1.p
0
>>> t1.p = 100    # change existing data member
>>> t1.p
100

>>> t1.z          # accessing non-existing member is an error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Track instance has no attribute 'z'
>>> t1.z = 0.5
>>> t1.z
0.5
>>> del t1.z
>>> t1.z
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: Track instance has no attribute 'z'
```

ラ 表示

ラ 表示 仕方 __str__と う関数&#x30

- **print**
- **str()**
- **"%s"**

でフォ マッ さ た場合

Track ラ 表示 仕方 設定 見 :

```
class Track(object):
   """Track represents a track with p, eta and phi components"""
   def __init__(self, p=0, eta=0, phi=0):
       """create a new track with p,eta,phi"""
       self.p = p
       self.eta = eta
       self.phi = phi

   def __str__(self):
       """nice string representation"""
       return "Track(p=%8.2f eta=%5.2f phi=%5.2f)" % (self.p, self.eta, self.phi)
```

```
>>> t1 = Track()
>>> print t1
Track(p=    0.00 eta= 0.00 phi= 0.00)
```

- メ ッド らデ タメ バ ア &#x3
  self.my_member う き (C++
  で this 省略可能)

特別なメ ッド

以下 う 数 カウ るため ラ

```
class Counter(object):
    def __init__(self, start=0):
        self.count = start

    def up(self, n=1):
        self.count += n

    def down(self, n=1):
        self.count -= n
```

ラ 以下 う 数 カウ る 使

```
>>> a = Counter(10)
>>> a.count
10
>>> a.up(2)
>>> a.count
12
```

で で ラ 表現と +
オ レ タ 加え み ょう

```
class AddCounter (Counter):
      def __repr__(self):
          return "AddCounter(%s)" % self.count
```

```
        def __add__(self, other):
            return AddCounter (self.count + other.count)
```

__repr__ __str__ と同様 文字列 返
__repr__
普通そ オブ ェ 作るため &#

今定義 たメ ッド 以下 う &#

```
>>> a = AddCounter (10)
>>> b = AddCounter (20)
>>> a
AddCounter (10)
>>> b
AddCounter (20)

>>> a+b                 # uses __add__ to create a new AddCounter instance
AddCounter (30)
>>> c = a+b             # ditto.
>>> c
AddCounter (30)

>>> c += AddCounter (2)  # there is no __iadd__ method defined on AddCounter,
>>> c                   # but python sees and then uses __add__:
AddCounter (32)          # c = c.__add__(AddCounter (2))

### corollary: beware that in this case, c is a new object (ie: if no __iadd__
### has been implemented) so its previous state will be somewhat lost:
>>> c = AddCounter (10)
>>> c
AddCounter (10)
>>> c.foo = 42
>>> c.foo
42
>>> c += AddCounter (3)
>>> c
AddCounter (13)
>>> c.foo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: AddCounter instance has no attribute 'foo'

### this doesn't happen with the following version of AddCounter:
class AddCounter (Counter):
        def __repr__(self):
            return "AddCounter(%s)" % self.count

        def __add__(self, other):
            return AddCounter (self.count + other.count)

        def __iadd__(self, other):
            self.count += other.count
            return self

>>> c = AddCounter (10)
>>> c
AddCounter (10)
>>> c.foo = 42
>>> c.foo
42
>>> c += AddCounter (3)
>>> c
AddCounter (13)
>>> c.foo
```

42

多重継承

```
class MuonTrack (Track, Muon):
    def __init__(self, p, eta, phi, muon_type):
        super(MuonTrack, self).__init__(p, eta, phi)
   self.muon_type = muon_type

    def __str__(self):
        s = "MuonTrack(%s muon_type=%s)" % (
       super(MuonTrack, self).__str__(),
       self.muon_type)
   return s
```

- 親 ラ 名 カッコ 中 羅列
- 子 ラ __init__ 中 ら親 ラ __init__ 呼び **Track.__init__(self, p, eta, phi)** う 直接呼ぶ 上記 う **super** 使 **super** で 両方 親 ラ メ ッドが&#x54
  - C++ う 自動的 呼ば るわけ&

プラ バ

- 全 メ バ が ブリッ
- 慣習的 ア コア(_)一つで&#x59
  C++ で言う所 "protected" なる : ラ 外 ら ア セ (可能だが)
- 同様 名前がア コア二&#x30
- ア コア二つで挟 る名&#x
  - __init__, __repr__, __add__, ...

ッ

- __slots__ 定義 た場合 動的 デ &#x30B

```
class Track(object):
      __slots__ = ('p', 'eta', 'phi')
      def __init__(self, p=0, eta=0, phi=0):
          self.p = p
          self.eta = eta
          self.phi = phi
```

- ア ラ で 打ち間違え るバ

```
>>> t = Track()
>>> t.rta = 2        # oops! 'rta' instead of 'eta'!!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Track' object has no attribute 'rta'
```

例外

例外 何ら エラ が起 った時
そ だとプ グラムが終了 &#12

通常例外 エラ っ 発生 る&#x3

```
raise RuntimeError ("something went terribly wrong")
```

- 例外 捕捉 る 以下 う `try` と `except` 使う

```
d = {}
try:
   # exceptions raised in this block are caught
   mistake = d['no-such-key']
except KeyError, e:
   # a KeyError exception was raised, and will be assigned to 'e'
   print e    # print the exception. uses KeyError.__str__
except AttributeError, e:
   # an AttributeError was raised
   print e
except Exception, e:
   # any other exception raised (that derives from class Exception)
   print e
else:
   # no exception was raised
   pass       # do nothing
```

組み込み 例外

- **KeyError**: `dict` 指定さ たキ がな
- **IndexError**: `list` や `tuple` 大きさ 超えた デッ が&#x630
- **AttributeError**: ラ 存在 な ア リビュ ア
- **NameError**: 値 設定さ な 変数へ ア &#x3
- **SyntaxError**: 構文が間違っ る
- **ImportError**: `import` で指定さ たモ ュ ルが見&
- **RuntimeError**: そ 他 例外

組み込み例外 フルリ :
http://docs.python.org/library/exceptions.html

例外 ラ 自作 る時 以下 う
Exception ラ 継承 る う る

```
class MyException (Exception):
     pass
```

```
>>> raise MyException ("foo")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
__main__.MyException: foo
```

モ ュ ル

モ ュ ルと (普通 関連 ある&#x6
.py

- 他 コ ド らモ ュ ル内 機&
  `import` る

```
import MyModule
```

- モ ュ ル 中 る ラ など &#x30

```
import MyModule
a = MyModule.SomeClass ()
```

- モ ュ ル名無 で ラ 名 そ &

```
from MyModule import SomeClass
a = SomeClass ()
```

- あるディレ リ **__init__.py**
  と うファ ルがある場合 &
  __init__.py
  ファ ル モ ュ ルが ポ さ &
  そ ディレ リ内 `python`
  ファ ル そ モ ュ ル サブ&#x3

```
from RecExConfig.RecFlags import Rec
```

- - **RecExConfig** がモ ュ ル
  - **RecFlags** がサブモ ュ ル
  - **Rec** **RecExConfig.RecFlags** オブ ェ

- **import** **$PYTHONPATH**
  と う環境変数 あるモ &#x30E5
- **import**
  っ そ モ ュ ル 中身が実&#x

標準モ ュ ル

- "batteries included"
  (電池が付属) 思想があ &#x591A

- **sys**: テム関連 モ ュ ル

```
>>> import sys
>>> sys.path      # list of paths to search for python modules
                  # i.e.: $PYTHONPATH + some system paths

>>> sys.modules   # dict of currently loaded modules

>>> sys.argv      # list of command line arguments
                  # sys.argv[0]  == name of the python script being run
```

```
    # sys.argv[1:] == arguments to the script
```

- **os**: OS関連 モ ュ ル

```
>>> import os
>>> os.listdir("/") # returns a list containing the names of the
                    # entries in the provided directory

>>> os.path.join("home", "foo") # joins 2 or more pathname components (cross-platform)

>>> os.path.exists("/tmp")    # tests whether a path exists
```

- **math**: 算術用 (**sin**, **cos**, **pow**, **sqrt**, **pi**, ...)
- **re**: 正規表現
- フルリ : http://docs.python.org/modindex.html

ファ ル入出&#x52

```
>>> f = open("myfile.txt", "w")    # open a file in write mode
>>> print &gt;&gt; f, 1, 2, 3, 4        # print a bunch of numbers into that file
>>> f.write('5 6 7 8')             # ditto
>>> f.write('9 10\n')              # ditto, but append a new-line
>>> f.flush()                      # commit any lingering buffer to file
>>> f.close()

>>> f = open("myfile.txt")         # default is to open file in read-only mode "r"
>>> for line in f:
...     print line
1 2 3 4

5 6 7 89 10

>>> f.seek(0)                      # rewind to beginning of file
>>> for line in f:
...     print repr(line)
'1 2 3 4\n'
'5 6 7 89 10\n'
```

ATLAS jobOptions ファ ル

最後 例と ATLAS jobOptions
(Athena コ フィグレ ファ ル) 見 &#12
次 ファ ル 明日 講習で使うH

https://svnweb.cern.ch/trac/atlasoff/browser/Generators/MC15JobOptions/trunk/share/DSID341xxx/MC15.341102.Pyt

文字列やリ 使っ ヒッグ &#3607

参考

http://docs.python.org

http://www.doughellmann.com/PyMOTW/

http://jacek.home.cern.ch/jacek/python-course/

http://pypi.python.org/pypi

http://www.scipy.org/

http://docs.scipy.org/doc/

http://matplotlib.sourceforge.net/index.html

http://docs.cython.org/

http://www.pytables.org/moin

http://h5py.alfven.org/docs/guide/quick.html

http://wlav.web.cern.ch/wlav/pyroot/

http://root.cern.ch/drupal/content/how-use-use-python-pyroot-interpreter

---

This topic: Main > AtlasJapanSoftwareTutorial16DecPython
Topic revision: r15 - 2017-01-23 - RyuSawada