Pretty Python 可爱的 Python

Chunting Gu, 2011/6/24

– KDIS Team Learning

Definition

Python is a

- strongly typed,
- dynamically, implicitly typed,
- object-oriented
- dynamic language.

Strongly Typed

强类型 **vs.** 若类型

Strong typing	**Weak typing**

`a = 2`
`b = '2'`
`concatenate(a, b)  # Type Error`
`add(a, b)  # Type Error`
`concatenate(str(a), b)  # '22'`
`add(a, int(b))  # 4`
	`a = 2`
	`b = '2'`
	
	`concatenate(a, b)  # '22'`
	`add(a, b)  # 4`

C, C++, Java, Python	PHP, JavaScript

```
>>> i, s = 1, '1'
>>> i + s
Traceback ...: ...
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

Dynamically Typed

动态类型

- Values have types but variables do not.

```
Variable                Values

  var         =         1
  var         =         "string"
  var         =         [1, 2, 3]
```

Variables are just names bound to values.

Implicitly Typed

隐式类型

- No type annotation.

C++0x

```
auto it = v.begin();
```

C#

```
var i = 1;
var s = "string";
```

Python

```
i, s = 1, "string"
```

```
def func(arg):
    pass
```

```
class A:
    def __init__(self):
        self.value = 0
```

Object-oriented

一切皆为对象

```
for attr in dir(1):
    print attr

__abs__
__add__
__and__
__cmp__
...
```

```
>>> def func():
...     """this is the doc of function func()"""
...     pass
>>> print func.__doc__
this is the doc of function f()
```

Dynamic Language

动态语言

- Eval（求值函数，e.g., eval(“1 + 2”) → 3)
- Object runtime alteration（运行时调整对象的方法和属性）
- Reflection（反射，自省）
- Closures（闭包）

> – 郑晖，《冒号课堂》，P. 134
> 动态语言能在运行中增加或改变数据结构、函数定义、对象行为或指令流程等。
> 如果说动态类型语言的动态体现在类型上，动态语言的动态则体现在结构和功能上。
> 静态语言也可以实现同样的效果，但既不方便也不自然。

Examples

- Swap values
- Reverse a string
- Loop in parallel over index and sequence items
- Loop through multiple lists

Swap Values

C++

```
t = a;
a = b;
b = t;
```

Python

```
a, b = b, a
```

More values:

```
a, b, c = b, c, a
```

Reverse String

C++

```cpp
std::string str = "123";
std::reverse(str.begin(), str.end());
```

Python

```python
"123"[::-1]
```

The slicing syntax:

```
[begin : end : step]
```

Loop In Parallel Over Index And Sequence Items

C++ for vector

```
for (size_t i = 0; i < V.size(); ++i)
```

C++ for list

```
for (list<T>::it = L.begin(), size_t i = 0;
    it != L.end();
    ++it, ++i)
```

Python

```
for idx, item in enumerate(L)
```

Loop Through Multiple Lists

```
a = ['a1', 'a2', 'a3']
b = ['b1', 'b2']
```

C++

```
for (it1 = a.begin(), it2 = b.begin();
    it1 != a.end() && it2 != b.end(); ++it1, ++it2)
```

Python

```
for x, y in map(None, a, b)
```

Or

```
for x, y in zip(a, b)
```

Higher-order Function 高阶函数

> Functions that act on or return other functions.

- Map
- Filter
- Reduce (fold)

Map

```
map(abs, [1, -2, 3, -4, 5])
# [1, 2, 3, 4, 5]
```

```
def sortedDictValues(d):
    keys = d.keys()
    keys.sort()
    return map(d.get, keys)

sortedDictValues({3: "3", 1: "1", 2: "2"})
# ['1', '2', '3']
```

Map (cont.)

Transpose two-dimensional arrays

```
[[1, 2, 3],              [[1, 4, 7],
 [4, 5, 6],       >>      [2, 5, 8],
 [7, 8, 9]]               [3, 6, 9]]
```

```
map(list, zip(*arr))
```

A note about zip:

```
zip(*arr) # →
zip([1, 2, 3], [4, 5, 6], [7, 8, 9]) # →
[(1, 4, 7), (2, 5, 8), (3, 6, 9)] # list of tuples
```

Filter

```
filter(lambda x: x > 0, [1, -2, 3, -4, 5])
# [1, 3, 5]
```

Find the intersection of two dictionaries

```
d1 = {1: "1", 2: "2", 3: "3"}
d2 = {3: "3", 4: "4", 5: "5"}
filter(d2.has_key, d1.keys())
# [3]
```

Reduce

Reduce in Python is the fold or accumulate in other languages.

```
reduce(operator.__add__, [1, 2, 3, 4, 5])
# 15
reduce(operator.__mul__, [1, 2, 3, 4, 5], 1)
# 120
```

Flatten a nested sequence:

```
reduce(list.__add__, [[1, 2], [3], [4, 5]])
# [1, 2, 3, 4, 5]
```

List Comprehension

列表推导, 列表内涵.

- A list comprehension is a syntactic construct for creating a list based on existinglists.
- It follows the form of the mathematical set-builder notation (set comprehension) as distinct from the use of map and filter functions.

Example:

```
[i for i in range(20) if i%2 == 0]
# [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Python Learnt List Comprehension From Haskell

Quicksort in Haskell

```
qsort [] = []
qsort x:xs = qsort [y | y <- xs, y < x]
             ++ [x]
             ++ qsort [y | y <-xs, y >= x]
```

Quicksort in Python

```
def qsort(L):
  if len(L) <= 1: return L
  return qsort([x for x in L[1:] if x < L[0]]) + L[0:1]
+\
    qsort([x for x in L[1:] if x >= L[0]])
```

Don’t use in product code!

Transpose Two-Dimensional Arrays

```
[[1, 2, 3],          [[1, 4, 7],
 [4, 5, 6],    >>     [2, 5, 8],
 [7, 8, 9]]           [3, 6, 9]]
```

Use map

```
map(list, zip(*arr))
```

Use list comprehension

```
[[r[col] for r in arr] for col in range(len(arr[0]))]
```

Find the Intersection of Two Dictionaries

Use filter

```
filter(d2.has_key, d1.keys())
```

Use list comprehension

```
[k for k in d1 if k in d2]
```

The End.

Thank You For Your Time.