

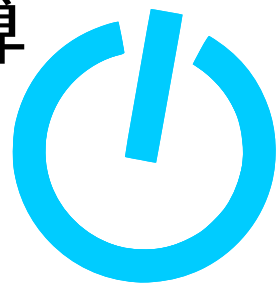
→ 需要虚拟机

Python Talk 9



函数式编程

- 函数式编程或称函数程序设计，又称泛函编程，是一种编程典范，它将电脑运算视为数学上的函数计算，并且避免使用程序状态以及易变物件。函数程式语言最重要的基础是 λ 演算（lambda calculus）。而且 λ 演算的函数可以接受函数当作输入（引数）和输出（传出值）。



Build-in Functions

- 多 → 一

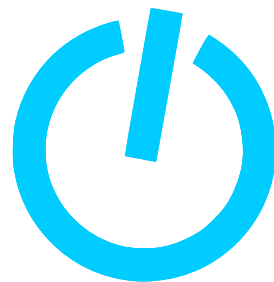
- all
- any
- len
- max
- min
- sum

- 多 → 多

- enumerate
- filter
- map
- reversed
- sorted
- zip

- 其他

- range
- slice
- yield



all, any, len

- all

- 所有元素都为真

- all([1, 2, 3])

- True

- all([1, 2, ''])

- False

- any

- 是否有任何元素为真

- any([0, ''])

- False

- any([0, 1])

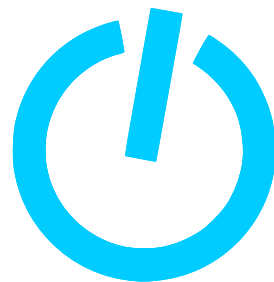
- True

- len

- 长度

- len([0, 'a', 9])

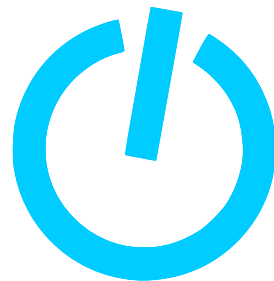
- 3



max, min, sum

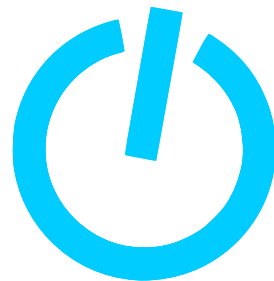
- max
 - 最大值
- min
 - 最小值
- sum
 - 总和

- max([1, 2, 3])
 - 3
- min([1, 2, 3])
 - 1
- sum([1, 2, 3])
 - 6



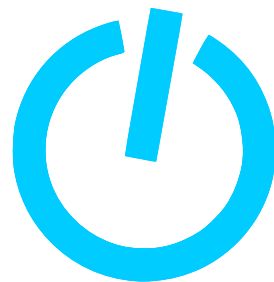
sorted, reversed

- sorted
 - 排序元素
- reversed
 - 反转元素
- `sorted([(9, 3), (4,), (2, 5)], key=lambda x: x[-1])`
 - `[(9, 3), (4,), (2, 5)]`



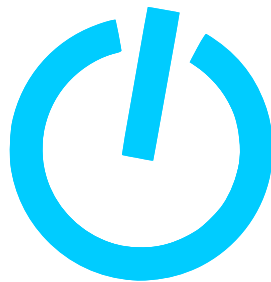
lambda 表达式

- **lambda** 参数列表： 函数返回值
- **lambda** x, y : $x ** y$
- $f = \text{lambda } x: x ** 2 + 3 * x + 1$
 - # 相当于 $f(x) = x^2 + 3x + 1$
 - # 优点：代码简洁
 - # 缺点：无法进行复杂运算



reversed 实践

- ```
>>> reversed([1, 2, 3])
<list_reverseiterator object at 0x123456789AB>
>>> # 这时返回的是一个 iter 的生成器，需要手动转换格式
>>> # 一般直接用 list 函数就可以了
>>> list(reversed([1, 2, 3]))
[3, 2, 1]
>>>
```





# enumerate

- enumerate 插入序号
  - ['a', 'b', 'c']
  - [
    - (0, 'a'),
    - (1, 'b'),
    - (2, 'c'),]
- for i, j in enumerate([1, 2, 3]) :
  - print(i, j)
- # 打印序号



# zip

- zip 可以拼接列表

- [1, 2, 3],

- [6, 7, 8]

- [

- (1, 6),

- (2, 7),

- (3, 8),

- ]

- A = [1, 2]

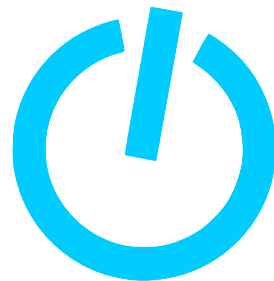
- B = ['a', 'b']

- for i, j in

- zip(a, b) :

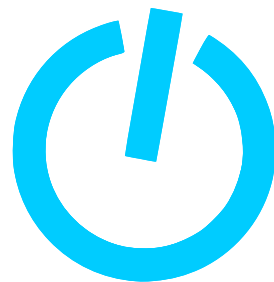
- print(i)

- print(j)



# map

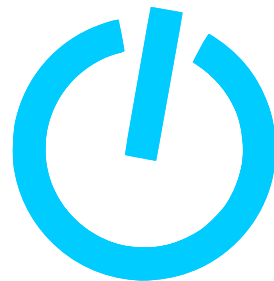
- map 可以将每个元素  
分别用函数执行
  - **lambda** x: x \*\* 2
  - [1, 2, 3, 4]
  - [1, 4, 9, 16]
- a = range(100)
- 尝试编写一个程序,  
用来求 a 的立方和
  - map
  - sum
  - **lambda**



# 立方和求解

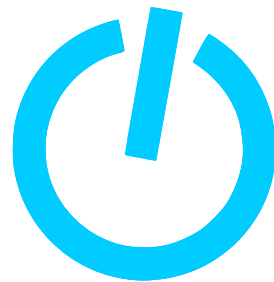
- `s = 0`
- `for i in range(a) :`
  - `s += i ** 3`
- `print(s)`

- `sum(map(lambda x: x**3, a))`
- `sum(`
  - `map(`
    - `lambda x: x**3,`
    - `a`
  - `)`
- `)`



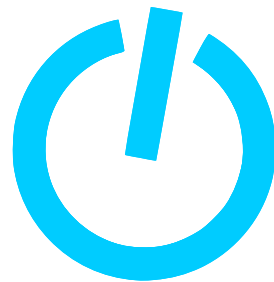
# filter

- `filter` 可筛选元素
  - **`lambda`** `x: x > 4`  
`[3, 4, 5, 6, 7]`
  - `[5, 6, 7]`
- `a = range(1000)`
- 求出 `a` 中三的倍数和七的倍数



# filter 实践

- `for i in a :`
  - `if i % 3 == 0 :`
    - `print(i)`
  - `if i % 7 == 0 :`
    - `print(i)`
- # 找出一处错误
- `filter(lambda x:  
x % 3 == 0 or x  
% 7 == 0, a)`



# range

- range 可以快速得到一个等差整数数列

```
- >>> range(1, 10, 2)
```

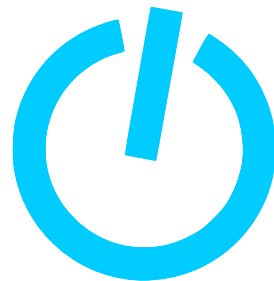
```
range(1, 10, 2)
```

```
>>> list(range(1, 10, 2))
```

```
[1, 3, 5, 7, 9]
```

```
>>> list(range(5, 3, -1)) # 反向
```

```
[5, 4]
```



# slice

- slice 即切片，和 `[a:b:c]` 相同

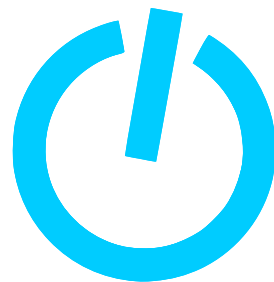
```
- >>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> a[2:7:2]
```

```
[3, 5, 7]
```

```
>>> a[slice(2, 7, 2)]
```

```
[3, 5, 7]
```

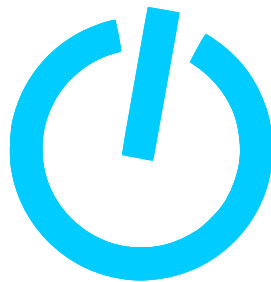




# yield

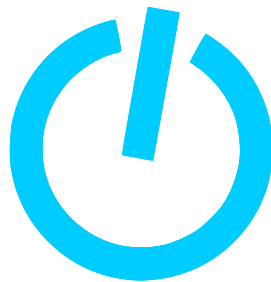
- `yield` 在函数中添加返回值，最后输出 `list`

```
- >>> def f(x):
... for i in range(x, 2 * x) :
... yield(i)
...
>>> list(f(3))
[3, 4, 5]
```



# 练习

- 以下程序可以干什么？
  - `sum(filter(lambda x: x % 3 == 1,  
map(lambda x: x ** 2, range(20))))`
  - # 需要匹配括号，见下页



# 括号匹配

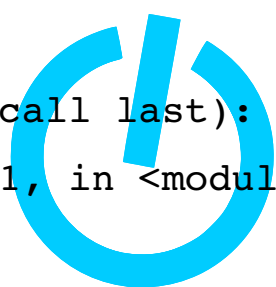
- `sum(`
  - `filter(`
    - `lambda x: x % 3 == 1,`
    - `map(lambda x : x ** 2, range(20))`
  - `)`
- `)`



# iter 和 next

```
• >>> a = [1, 2]
>>> b = a.__iter__()
>>> b.__next__()
1
>>> b.__next__()
2
>>> b.__next__()
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
StopIteration
>>>
```

```
• >>> a = [1, 2]
>>> b = iter(a)
>>> next(b)
1
>>> next(b)
2
>>> next(b)
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
StopIteration
>>>
```



# 感谢参加此次活动

