

# Python语言项目教学03

陈斌

北京大学地球与空间科学学院

2018.10.26

# 目录

- Python语言要件概览
- 运算和赋值语句
- 条件语句if
- 循环语句while/for
- 上机练习
- 函数定义
- 函数参数和调用
- 上机练习



# Python语言的几个要件

## 数据对象和组织

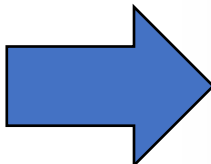
- 对现实世界实体和概念的抽象
- 分为简单类型和容器类型
- 简单类型用来表示值
  - 整数int、浮点数float、复数complex、逻辑值bool、字符串str
- 容器类型用来组织这些值
  - 列表list、元组tuple、集合set、字典dict
- 数据类型之间几乎都可以转换

## 赋值和控制流

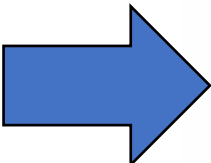
- 对现实世界处理和过程的抽象
- 分为运算语句和控制流语句
- 运算语句用来实现处理与暂存
  - 表达式计算、函数调用、赋值
- 控制流语句用来组织语句描述过程
  - 顺序、条件分支、循环
- 定义语句也用来组织语句，描述一个包含一系列处理过程的计算单元
  - 函数定义、类定义

# 运算语句：表达式、函数调用和赋值

- 各种类型的数据对象，可以通过各种运算组织成复杂的表达式
- 调用函数或者对象，也可以返回数据，所有可调用的事物称为 callable
  - 调用函数或者对象，需要在其名称后加圆括号，如果有参数，写在圆括号里
  - 不加圆括号的函数或者对象名称仅是表示自己，不是调用
- 将表达式或者调用返回值传递给变量进行引用，称为赋值



```
>>> 12 * 34.5 + 23.4
437.4
>>> ('abc' + '123') * 3
'abc123abc123abc123'
>>>
>>> import math
>>> math.sqrt(12)
3.4641016151377544
>>> math.sqrt
<built-in function sqrt>
>>>
>>> n = 12 * 34
>>> n
408
>>> p2 = math.sqrt(2)
>>> p2
1.4142135623730951
>>> pfg = math.sqrt
>>> pfg
<built-in function sqrt>
>>> pfg(2)
1.4142135623730951
```



# 赋值语句的小技巧

- 级联赋值语句

- `x = y = z = 1`

- 多个变量分解赋值

- `a, b = ['hello', 'world']`

- 变量交换

- `a, b = b, a`

- 自操作

- `i += 1`

- `n *= 45`

```
>>> x = y = z = 1
>>> x, y, z
(1, 1, 1)
>>> a, b = ['hello', 'world']
>>> a
'hello'
>>> b
'world'
>>>
>>> a, b = b, a
>>> a
'world'
>>> b
'hello'
>>>
>>> a += 'cup'
>>> a
'worldcup'
```

# 控制流语句：条件if

- 条件语句

- `if` <逻辑条件>:
- <语句块>
- `elif` <逻辑条件>: #可以多个`elif`
- <语句块>
- `else`: #仅1个
- <语句块>

- 各种类型中某些值会自动被转换为False，其它值则是True:

- `None`, `0`, `0.0`, `''`,
- `[]`, `()`, `{}`, `set()`

```
>>> a = 12
>>> if a > 10:
        print ("Great!")
elif a > 6:
        print ("Middle!")
else:
        print ("Low!")

Great!
```

# 控制流语句：while循环

- 条件循环while
  - while <逻辑条件>:
    - <语句块>
    - break #跳出循环
    - continue #略过余下循环语句
    - <语句块>
  - else: #条件不满足退出循环，则执行
  - <语句块>
- else中可以判断循环是否遭遇了break

```
>>> n = 5
>>> while n > 0:
        n = n - 1
        if n < 2:
            break
        print (n)
```

4  
3  
2

```
>>> n = 5
>>> while n > 0:
        n = n - 1
        if n < 2:
            continue
        print (n)
else:
    print ('END!')
```

4  
3  
2  
END!



# 控制流语句：for循环

- 迭代循环for:

- for <变量> in <可迭代对象>:
- <语句块>
- break #跳出循环
- continue #略过余下循环语句
- else: #迭代完毕，则执行
- <语句块>

- 可迭代对象有很多类型

- 象字符串、列表、元组、字典、集合等
- 也可以有后面提到的生成器、迭代器等

```
>>> for n in range(5):  
        print (n)
```

```
0  
1  
2  
3  
4
```

```
>>> alist = ['a', 123, True]  
>>> for v in alist:  
        print (v)
```

```
a  
123  
True
```

```
>>> adic = {'name': 'Tom', 'age': 18, 'gender': 'Male'}  
>>> for k in adic:  
        print (k, adic[k])
```

```
name Tom  
age 18  
gender Male
```

```
>>> for k, v in adic.items():  
        print (k, v)
```

```
name Tom  
age 18  
gender Male
```



# 迭代的小技巧：zip()函数

- 由于for循环的迭代是对容器中的数据项进行枚举，但不带序号或下标
  - 有时候我们需要数据项的序号
  - 另外，有时我们需要并行迭代两组一一对应的数据项
- 使用zip()打包函数可以使代码更加简洁
  - zip(list1, list2, list3...)返回由每个列表对应位置的数据项构成元组的列表。

```
weekdays = ['Sun', 'Mon', 'Tue', 'Wed', \
             'Thu', 'Fri', 'Sat']

for n, s in zip(range(7), weekdays):
    print (n, s)

for n in range(7):
    print (n, weekdays[n])

n = 0
while n < 7:
    print (n, weekdays[n])
    n = n + 1

drinks = ['coffee', 'tea', 'beer', \
          'water', 'milk', 'coca-cola', 'none']

for w, d in zip(weekdays, drinks):
    print ('drink', d, 'in', w)
```

```
0 Sun
1 Mon
2 Tue
3 Wed
4 Thu
5 Fri
6 Sat
```

```
drink coffee in Sun
drink tea in Mon
drink beer in Tue
drink water in Wed
drink milk in Thu
drink coca-cola in Fri
drink none in Sat
```

# 上机练习

① 给定 $n$ ，计算 $1+2!+3!+\dots+n!$ 的值

② 给定 $y$ 和 $m$ ，计算 $y$ 年 $m$ 月有几天？

注意闰年定义

③ 给定字符串 $s$ 和数字 $n$ ，打印把字符串 $s$ 向右移动 $n$ 位的新字符串

例如`abcd`和`1`，返回`dabc`

例如`mnbo1`和`2`，返回`olmnb`

④ 给定一个英文数字字符串，打印相应阿拉伯数字字符串

例如：`one-four-five-nine`

返回：`1459`

# 函数function

- 函数用来对具有明确功能的代码段命名，以便复用 (reuse)
- 定义函数：def语句；
  - def <函数名> (<参数表>):
  - <缩进的代码段>
  - return <函数返回值>
- 调用函数：<函数名> (<参数>)
  - 注意括号！
  - 无返回值：<函数名> (<参数表>)
  - 返回值赋值：v = <函数名> (<参数表>)

```
1  def sum_list(alist): # 定义一个带参数的函数
2      sum_temp = 0
3      for i in alist:
4          sum_temp += i
5      return sum_temp # 函数返回值
6
7
8  print(sum_list) # 查看函数对象sum_list
9
10 my_list = [23, 45, 67, 89, 100]
11 # 调用函数，将返回值赋值给my_sum
12 my_sum = sum_list(my_list)
13 print("sum of my list:%d" % (my_sum,))
```

```
<function sum_list at 0x10067a620>
sum of my list:324
```

# 函数定义中的代码块

- 由于函数定义def语句仅仅是把代码块“打包封装”
- def语句执行的时候，代码块并不会被执行
- 所以，在执行def语句的时候，除非语句块中包含了明显的语法错误
- Python解释器是不会检查语句块中其它错误的。

func\_star2.py

```
1  import turtle
2
3
4  def star(size, color):
5      t.color(color)
6      t.begin_fill()
7      for i in range(5):
8          t.forward(size)
9          t.left(72)
10         t.forward(size)
11         t.right(144)
12     t.end_fill()
13
14
15 t = turtle.Turtle()
16
17 star(20, 'red')
```

并不会出现  
“t未定义”  
的错误

# 定义函数的参数： 固定参数／可变参数

- 定义函数时，参数可以有两种；
- 一种是在参数表中写明参数名key的参数，固定了顺序和数量
  - `def func(key1, key2, key3...):`
  - `def func(key1, key2=value2...):`
- 一种是定义时还不知道会有多少参数传入的可变参数
  - `def func(*args):` #不带key的多个参数
  - `def func(**kwargs):` #key=value形式的多个参数

```
16 def func_test(key1, key2, key3=23):
17     print("k1=%s,k2=%s,k3=%s" % (key1, key2, key3))
18
19
20 print("====func_test")
21 # 没有传入key3, 用了缺省值
22 func_test('v1', 'v2')
23 # 传入了key3
24 func_test('ab', 'cd', 768)
25 # 使用参数名称就可以不管顺序
26 func_test(key2='KK', key1='K')
```

```
====func_test
k1=v1,k2=v2,k3=23
k1=ab,k2=cd,k3=768
k1=K,k2=KK,k3=23
```



# 定义函数的参数： 固定参数／可变参数

```
29 # 可以随意传入0个或多个无名参数
30 def func_test2(*args):
31     for arg, i in zip(args, range(len(args))):
32         print("arg%d=%s" % (i, arg))
33
34
35 print("====func_test2")
36 func_test2(12, 34, 'abcd', True)
```

```
====func_test2
arg0=12
arg1=34
arg2=abcd
arg3=True
```

```
39 # 可以随意传入0个或多个带名参数
40 def func_test3(**kwargs):
41     for key, val in kwargs.items():
42         print("%s=%s" % (key, val))
43
44
45 print("====func_test3")
46 func_test3(myname="Tom", sep="comma", age=23)
```

```
====func_test3
sep=comma
age=23
myname=Tom
```

# 调用函数的参数：位置参数／关键字参数

- 调用函数的时候，可以传进两种参数；
- 一种是没有名字的位置参数
  - `func(arg1, arg2, arg3...)`
  - 会按照前后顺序对应到函数参数
- 一种是带key的关键字参数
  - `func(key1=arg1, key2=arg2...)`
  - 由于指定了key，可不按顺序对应
- 如果混用，所有位置参数必须在前，关键字参数必须在后

```
16 def func_test(key1, key2, key3=23):
17     print("k1=%s,k2=%s,k3=%s" % (key1, key2, key3))
18
19
20     print("====func_test")
21     # 没有传入key3, 用了缺省值
22     func_test('v1', 'v2')
23     # 传入了key3
24     func_test('ab', 'cd', 768)
25     # 使用参数名称就可以不管顺序
26     func_test(key2='KK', key1='K')
```

```
====func_test
k1=v1,k2=v2,k3=23
k1=ab,k2=cd,k3=768
k1=K,k2=KK,k3=23
```



# 函数小技巧：map()函数

- 有时候，需要对列表中每个元素做一个相同的处理，得到新列表
  - 例如所有数据乘以3
  - 例如所有字符串转换为整数
  - 例如两个列表对应值相加
- `map(func, list1, list2....)`
  - 函数func有几个参数，后面跟几个列表

```
num = [10, 20, 40, 80, 160]
lst = [2, 4, 6, 8, 10]
def mul3(a):
    return a * 3

print (list( map(mul3, num) ))

def atob(a, b):
    return a + 1.0/b

print (list( map(atob, num, lst) ))
```

```
[30, 60, 120, 240, 480]
[10.5, 20.25, 40.166666666666664, 80.125, 160.1]
```

# 函数小技巧：匿名函数lambda

- 有时候，函数只用一次，其名称也就不重要，可以无需费神去def一个
- Lambda表达式可以返回一个匿名函数
  - `lambda <参数表>:<表达式>`

```
num = [10, 20, 40, 80, 160]
lst = [2, 4, 6, 8, 10]
def mul3(a):
    return a * 3

print (list( map(mul3, num) ))

def atob(a, b):
    return a + 1.0/b

print (list( map(atob, num, lst) ))

print (list( map(lambda a:a * 3, num)))
print (list( map(lambda a,b:a+1.0/b, num, lst)))
```

# 上机练习：函数定义

- 水仙花数判定：创建一个函数，接受一个参数 $n(n \geq 100)$ ，判断这个数是否为水仙花数
  - 即满足如果这个数为 $m$ 位数，则每个位上的数字的 $m$ 次幂之和等于它本身，例如 $1^3 + 5^3 + 3^3 = 153$ ， $1^4 + 6^4 + 3^4 + 4^4 = 1634$ ），返回True或者False。
- 创建一个函数，接受一个参数 $\text{max}(\text{max} \geq 1000)$ ，调用上题编写的判断函数，求100到 $\text{max}$ 之间的水仙花数。
- 创建一个函数，接受两个字符串作为参数，返回两个字符串字符集合的并集。
  - 如接受的两个字符串为"abc"和"bcd"，返回`set(['a', 'b', 'c', 'd'])`。