

第 3 次课 函数与分支

Python 科学计算

周吕文

宁波大学，机械工程与力学学院

2024 年 9 月 1 日



Notes

提要

1 函数

2 分支

3 测试

Notes

Python 中的多种函数

数学函数

```
>>> from math import *
>>> x = pi/2
>>> y = sin(x)*log(x)
```

其它函数

```
>>> NBU = ['N', 'B', 'U', 315211]
>>> n = len(NBU)
>>> integers = list(range(1, n, 2))
```

用点符号调用的函数（也称方法）

```
>>> i = NBU.index(315211)
>>> NBU.append(818)
```

Notes

函数很重要

什么是函数

输入（参数） → 函数：一段代码 → 输出（返回值）

函数的用处

- 函数可以帮助组织程序，增加程序的可读性、重用性，使程序更短并易于扩展。
- 我们可以自行定义函数，实现我们自己的功能。

Notes

公式 $f(C) = \frac{9}{5}C + 32$ 可以用以下 Python 函数实现 c2f.py

```
def F(C):  
    return 9/5*C + 32  
  
print('F = ', F(5))
```

```
>>>  
F = 41.0
```

注意

- 函数头: **def** 开始, 然后是函数名 F 和参数 C, 最后以冒号结尾
- 函数体: 缩进的一组语句
- 可以在函数体任何地方通过 **return** 结束函数的运行并带回返回值

函数只有在被调用时才执行

```
def F(C):  
    return 9/5*C + 32  
  
a = 10  
  
F1 = F(a)  
print('F1 = %.1f' % F1)  
  
temp = F(15.5)  
print('temp = %.1f' % temp)  
  
sumt = F(10) + F(20)  
print('sumt =', sumt)  
  
Fs = [F(C) for C in [0, 20]]  
print('Fs =', Fs)
```

Global frame

F(C) function			
a	10	F1	50.0
temp	59.9	sumt	118.0
Fs [32.0, 68.0]			

In function F

C		return
---	--	--------

IDLE Shell

```
>>>  
F1 = 50.0  
temp = 59.9  
sumt = 118.0  
Fs = [32.0, 68.0]
```

函数可以有多个参数

公式 $y(t) = v_0t - \frac{1}{2}gt^2$ 可写成带两个参数的函数 ball.py

```
def yfunc(t, v0):  
    g = 9.81  
    return v0*t - 0.5*g*t**2  
  
y = yfunc(0.1, 6); print(y)  
  
y = yfunc(0.1, v0=6); print(y)  
  
y = yfunc(t=0.1, v0=6); print(y)  
  
y = yfunc(v0=6, t=0.1); print(y)
```

```
>>>  
0.55095  
0.55095  
0.55095  
0.55095
```

函数参数是局部变量

ball_local.py

```
def yfunc(t, v0):  
    g = 9.81  
    return v0*t - 0.5*g*t**2  
  
g, v0, t = 10, 5, 0.6  
y = yfunc(t, 3)  
print('y = %.1f' % y)  
print('g = %.1f' % g)
```

```
>>>  
y = 0.0  
g = 10.0
```

局部变量和全局变量

- 在函数 **yfunc** 中, **t**、**v0**、**g** 都是局部变量, 在函数不可见, 而且在函数执行完后被销毁
- 函数外的 **v0**、**t**、**y** 是全局变量, 在程序的任何地方都可见

函数可以访问全局变量

函数访问的全局变量必须在函数调用之前定义

```
>>> def yfunc(t):
...     g = 9.81
...     return v0*t - 0.5*g*t**2
...
>>> yfunc(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in yfunc
NameError: name 'v0' is not defined

>>> v0 = 0.6
>>> yfunc(3)
-42.345000000000006
```

Notes

局部变量会屏蔽同名的全局变量

ball_local_global.py

```
def yfunc(t):
    print('1. 局部变量 t:', t)
    g = 9.81
    t = 0.1
    print('2. 局部变量 t:', t)
    return v0*t - 0.5*g*t**2

t = 0.6
v0 = 2
print(yfunc(t))
print('1. 全局变量 t: %.1f\n' % t)

print(yfunc(0.3))
print('2. 全局变量 t: %.1f\n' % t)
```

>>>

```
1. 局部变量 t: 0.6
2. 局部变量 t: 0.1
0.15095
1. 全局变量 t: 0.6

1. 局部变量 t: 0.3
2. 局部变量 t: 0.1
0.15095
2. 全局变量 t: 0.6
```

Notes

通过声明为 global 可以在函数中修改全局变量

ball_global.py

```
def yfunc(t):
    g = 9.81
    global v0      # 若注释掉这一行，输出结果会变成什么？
    v0 = 9
    return v0*t - 0.5*g*t**2

v0 = 2
print('1. v0:', v0)

yfunc(0.8)
print('2. v0:', v0)
```

>>>

```
1. v0: 2
2. v0: 9
```

Notes

函数可以返回多个值

ball_multi_return.py

```
def yfunc(t, v0):
    g = 9.81
    y = v0*t - 0.5*g*t**2
    dydt = v0 - g*t # 计算 y 对 t 的导数
    return y, dydt # 实际上，返回多个值时返回的一个元组

print(yfunc(0.6, 3))

position, velocity = yfunc(0.6, 3)
print(position, velocity)
```

>>>

```
(0.034199999999999786, -2.886)
0.034199999999999786 -2.886
```

Notes

例子：计算求和方程

$$\ln(1+x) \approx L(x; n) = \sum_{i=1}^n \frac{1}{i} \left(\frac{x}{1+x} \right)^i, \quad x \geq 1, n \in \mathbb{N}$$

实现函数 $L(x; n)$ 的 python 代码

Insum.py

```
def L(x, n):
    s = 0
    for i in range(1, n+1):
        s += 1/i * (x/(1+x))**i
    return s
```

```
from math import log as ln
x = 5
print(L(x, 10))
print(L(x, 100))
print(ln(1+x))
```

```
>>>
1.7368363532538857
1.7917594686571028
1.791759469228055
```

Notes

例子：计算求和方程

计算 $L(x, n)$ 时同时计算误差并返回

Insum_error.py

```
def L2(x, n):
    s = L(x, n) # 调用之前定义的 L(x, n) 函数
    first_neglected_term = (1/(n+1))*(x/(1+x))**(n+1)
    from math import log
    exact_error = log(1+x) - s
    return s, first_neglected_term, exact_error
```

```
x = 1.2; n = 100
value, approx_error, exact_error = L2(x, n)
print('%4f %2e %2e'%(value, approx_error, exact_error))
```

```
>>>
0.7885 2.56e-29 3.33e-16
```

Notes

不需要返回值的函数

函数没有 return 语句，会自动 return None

return_none.py

```
def somefunc(obj):
    print('函了个寂寞: ', obj)

return_value = somefunc(666)
print(return_value)
```

```
>>>
函了个寂寞: 666
None
```

例子：输出 n 和 $L(x, n)$ 对应表

Insum_table.py

```
def table(x):
    print('%4s L(n,%g)' % ('n', x))
    for n in [1, 10, 100]:
        v = L(x, n)
        print('%4d %7.4f' % (n, v))
```

table(10)

```
>>>
n L(n,10)
1 0.9091
10 2.1791
100 2.3979
```

Notes

默认参数 (default argument)

默认参数：函数定义时形如 name = value 的参数

```
>>> def somefunc(arg1, arg2, kwarg1=True, kwarg2=0):
...     print(arg1, arg2, kwarg1, kwarg2)
...
>>> somefunc('Hello', [1,2])
Hello [1, 2] True 0
>>> somefunc('Hello', [1,2], kwarg2='Hi')
Hello [1, 2] True Hi
>>> somefunc('Hello', [1,2], kwarg2='Hi', kwarg1=6)
Hello [1, 2] 6 Hi

>>> # 若调用函数时都给出参数名 (关键字)，则参数顺序无所谓
>>> somefunc(kwarg2='Hello', arg1='Hi', kwarg1=6, arg2=[2])
Hi [2] 6 Hello
```

Notes

实现涉及 1 个变量但有多个参数的函数

考虑 t 的函数 $f(t) = Ae^{-at} \sin(\omega t)$, 其中 A 、 a 、 ω 为默认参数

```
>>> from math import pi, exp, sin
>>> def f(t, A=1, a=1, omega=2*pi):
...     return A*exp(-a*t)*sin(omega*t)
...

>>> f(0.2)                                >>> f(0.2, 1, 3)
0.778659217806053                        0.5219508827258282
>>> f(0.2, omega=1, A=2.5)                >>> f(t=0.2, A=9)
0.40664172703834794                      7.007932960254476
>>> f(A=5, a=0.1, omega=1, t=1.3)
4.230480200204721
>>> f(t=0.2, 9)
File "<stdin>", line 1
    f(t=0.2, 9)
    ^
SyntaxError: positional argument follows keyword argument
```

Notes

文档字符串（Doc Strings）

在函数头后，用三引号字符串说明函数功能、参数等 doc_strs.py

```
def C2F(C):
    """将摄氏度转化为华氏度"""
    return 9/5*C + 32

def line(x0, y0, x1, y1):
    """
    通过直线上的两点计算直线  $y = ax + b$  的斜率和截距。
    x0, y0: 直线上的一个点 (floats)
    x1, y1: 直线上另一个点 (floats)
    返回: a, b (floats)
    """
    a = (y1 - y0)/(x1 - x0)
    b = y0 - a*x0
    return a, b
```

Notes

参数是输入，返回值是输出

函数的输入和输出可包含三类

- 仅输入数据，例如 i1, i2, i3, i6
- 输入/输出数据，例如 io4, io5, io7
- 仅输出数据，例如 o1, o2, o3

```
def somefunc(i1, i2, i3, io4, io5, i6=value1, io7=value2):
    # 修改 io4, io5, io7
    # 计算 o1, o2, o3
    return o1, o2, o3, io4, io5, io7
```

Notes

主程序：函数外的一组语句

```
in_main.py
from math import *           # 主程序语句

def f(x):                    # 主程序语句
    e = exp(-0.1*x)
    s = sin(6*pi*x)
    return e*s

x = 2                         # 主程序语句
y = f(x)                      # 主程序语句
print('f(%g)=%g' % (x, y))   # 主程序语句
```

- 程序从主程序第一条语句开始执行，从上到下，逐条执行
- def 语句定义了一个函数，该函数被调用前，其中的语句不会执行

Notes

1 创建生成列表的函数

文件名: mklist.py

编写函数 `mklist`, 根据起始值、结束值和增量, 创建等差数列列表。要求默认增量为 1, 且要有文档字符串。例如

`mklist(2, 5, 0.5)` 将返回 `[2.0, 2.5, 3.0, 3.5, 4.0, 4.5]`

2 计算球面距离

文件名: distance.py

半径为 R 的球面上的两点距离可由以下公式计算

$$d_{i,j} = R \cdot \arccos [\sin \phi_i \sin \phi_j + \cos \phi_i \cos \phi_j \cos(\theta_i - \theta_j)]$$

其中 ϕ_i 和 θ_i 表示第 i 个节点的纬度 (北正南负) 和经度 (东正西负)。

- 编写函数 `distance`, 根据给定两点经纬度, 计算球面距离。
- 已知地球半径为 6371 km, 北京和纽约的经纬度坐标分别为 (39.90° N, 116.41° E) 和 (40.71° N, 74.01° W), 求两点球面距离。

Notes

函数作为参数的函数

有时需要将一个函数作为另一个函数的参数, 如求数值二阶导数

$$f''(x) \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2}$$

diff2nd.py

```
def diff2(f, x, h=1E-6): # 参数 f 为函数
    r = (f(x-h) - 2*f(x) + f(x+h))/h**2
    return r

def g(t): return t**(-6)

for k in range(1,11):
    h = 10**(-k)
    g2 = diff2(g, 1, h)
    print('h=%0e: %.2f' % (h, g2))
```

>>>

```
h=1e-01: 44.62
h=1e-02: 42.03
h=1e-03: 42.00
h=1e-04: 42.00
h=1e-05: 42.00
h=1e-06: 42.00
h=1e-07: 41.94
h=1e-08: 47.74
h=1e-09: -666.13
h=1e-10: 0.00
```

Notes

函数作为参数的函数

问题: 当 h 很小时结果错误

- 理论上 h 越小, 得到的结果会越精确。
- 但当 $h < 10^{-8}$ 时, 结果完全错了。
- 更确切的说, 数学结果在 $h < 10^{-4}$ 时就不再成立。

原因: 舍入误差导致了错误结果

- h 越小时, 舍入同等大小的数所占比例越大
- h 很小的误差在计算中被放大了

补救方法

- 使用精确度更高的浮点数。Python 有一种 (速度较慢的) 浮点变量 (`decimal.Decimal`), 可以处理任意位数的数字。
- 其实不用太担心, 一般输入也不会使用那么小的数。

Notes

Lambda 函数: 更紧凑的函数定义形式

```
def somefunc(a1, a2, ...):
    return some_expression
```

⇕

```
somefunc = lambda a1, a2, ...: some_expression
```

Lambda 函数实例: $f(x) = x^2 - 1$

```
>>> f = lambda x: x**2 - 1
>>> f(2)
3
```

Lambda 函数实例: $f(t) = t^{-6}$ 在 $t = 1$ 处的数值二阶导数

```
# 调用之前定义的函数 diff2(f, x, h=1E-6)
dgdtd = diff2(lambda t: t**(-6), 1)
print('%.6f' % dgdtd)
```

>>>

42.000736

Notes

课堂练习

3 创建计算小球位置的匿名函数 文件名: ball_lambda.py

编写 lambda 函数, 根据以下公式计算以速度 v 上抛的小球 t 时刻的位置:

$$y(t) = vt - \frac{1}{2}gt^2, \quad g = 9.8 \text{ m/s}^2$$

4 梯形数值积分 文件名: trapezint.py

函数 $f(x)$ 在区间 $[a, b]$ 上的积分可以使用梯形近似法计算:

$$\int_a^b f(x) dx = \sum_{i=0}^{n-1} \frac{1}{2} h [f(x_i) + f(x_{i+1})], \quad h = \frac{b-a}{n}, \quad x_i = a + ih$$

编写函数 trapezint(f, a, b, n) 实现上述积分法, 并计算以下情况:

- $f = \sin, a = 0, b = \pi/2, n = 10$
- $f = x^2, a = 0, b = 3, n = 100$

Notes

提要

1 函数

2 分支

3 测试

Notes

If 分支结构

有时需要根据某个条件 (如 x 的值) 选择不同的操作

$$f(x) = \begin{cases} \sin x & 0 \leq x \leq \pi \\ 0 & \text{otherwise} \end{cases}$$

```
from math import sin, pi
def f(x):
    if 0 <= x <= pi:
        return sin(x)
    else:
        return 0

print(f(pi/2))
print(f(2*pi))
```

In function f

x pi/2 | return 1.0

IDLE Shell

```
>>>
1.0
0
>>>
```

Notes

If 分支结构的一般形式

单分支结构

```
if <布尔条件>:
    <语句块, 条件为真时执行>

<分支结构外语句>
```

二分支结构

```
if <布尔条件>:
    <语句块, 条件为真时执行>
else:
    <语句块, 条件为假时执行>

<分支结构外语句>
```

多分支结构

```
if <布尔条件 1>:
    <语句块>
elif <布尔条件 2>:
    <语句块>
    :
elif <布尔条件 n>:
    <语句块>
else:
    <语句块>

<分支结构外语句>
```

Notes

多分支举例

branch_fun.py

```
def N(x):
    if x < 0:
        return 0
    elif 0 <= x < 1:
        return x
    elif 1 <= x < 2:
        return 2 - x
    elif x >= 2:
        return 0
print('N(-0.2) =', N(-0.2))
print('N( 0.5) =', N( 0.5))
print('N( 1.2) =', N( 1.2))
print('N( 3.8) =', N( 3.8))
```

分段函数

$$N(x) = \begin{cases} 0 & x < 0 \\ x & 0 \leq x < 1 \\ 2 - x & 1 \leq x < 2 \\ 0 & x \geq 2 \end{cases}$$

```
>>>
N(-0.2) = 0
N( 0.5) = 0.5
N( 1.2) = 0.8
N( 3.8) = 0
>>>
```

周启龙 宁波大学 分支 2024 年 9 月 1 日 29 / 36

Notes

单行 If 语句

```
if <布尔条件>:
    variable = value1
else:
    variable = value2
```

⇕

```
variable = (value1 if <布尔条件> else value2)
```

实例：分段函数

```
>>> from math import sin, pi
>>> def f(x):
...     return (sin(x) if 0 <= x <= pi else 0)
...
>>> f(pi/2)                >>> f(2*pi)
1.0                        0
```

周启龙 宁波大学 分支 2024 年 9 月 1 日 30 / 36

Notes

课堂练习

5 学生成绩评级

文件名：grade.py

编写一个函数，根据输入的分输出等级：

- 90-100 A；80-89 B；70-79 C；60-69 D；小于 60 E。
- 随机生成一个包含 10 个学生成绩的列表，并调用函数计算等级。

6 判断某年是否为闰年

文件名：isleap.py

编写函数，根据输入年份判断该年是否为闰年。满足以下两个条件的整数才可以称为闰年：

- 普通闰年：能被 4 整除但不能被 100 整除（如 2004 年就是普通闰年）
- 世纪闰年：能被 400 整除（如 2000 年是世纪闰年，而 1900 年不是）

调用该函数进行验证其正确性：1994, 1996, 2000, 2024。

周启龙 宁波大学 分支 2024 年 9 月 1 日 31 / 36

Notes

提要

Notes

- 1 函数
- 2 分支
- 3 测试


```
double_test_error.py
def double(x): return x**2 # 应为 2*x, 假装写错了

def test_double(): # 测试函数: 对 double(x) 函数进行测试
    x = 4; computed = double(x); expected = 8
    msg = '测试输出 %s, 期望输出 %s' % (computed, expected)
    assert computed == expected, msg

test_double()

AssertionError: 测试输出 16, 期望输出 8
```

测试函数编写规则

- 测试函数命名通常以 test_ 开始, 没有参数
- 必须有 assert 语句, 测试通过无输出, 否则引发 AssertionError
- 测试信息 msg 可选, 测试不通过时输出 msg

Notes

```
double_multi_test.py
def double(x): # 函数
    return 2*x

def test_double(): # 测试函数: 对 double(x) 函数进行测试
    tol = 1E-14
    x_values = [3, 7, -2, 0, 4.5]
    expected_values = [6, 14, -4, 0, 9]
    for x, expected in zip(x_values, expected_values):
        computed = double(x)
        msg = '%s != %s' % (computed, expected)
        assert abs(expected - computed) < tol, msg

test_double() # 测试通过则什么也不输出!

>>>
```

Notes

为什么遵循这些规则来写测试程序?

- 通常以 test_ 开始: 方便辨认这是要测试谁
- 测试框架可以方便的运行你的所有测试程序
- 已成为良好编程习惯的标准
- test_double 只是测试一个函数, 这种测试也叫单元测试。若每个单元都通过测试, 那整个程序也就差不多了。

测试程序很难吗?

- 看似复杂, 原理很简单: 构造某个已知答案的问题, 调用函数计算结果, 比较答案和计算结果。
- 通过测试函数没有输出, 需要时可在开发过程中插入一些输出。
- 实际系统开发时, 有时候甚至建议先写测试程序, 再写程序。

Notes

7 测试函数 文件名: trapezint.py

函数 $f(x)$ 在区间 $[a, b]$ 上的积分可以使用梯形近似法计算:

$$\int_a^b f(x) dx = \sum_{i=0}^{n-1} \frac{1}{2} h [f(x_i) + f(x_{i+1})], \quad h = \frac{b-a}{n}, \quad x_i = a + ih$$

编写函数 trapezint(f, a, b, n) 实现上述积分法。针对函数 trapezint 编写测试函数, 对以下几种情况进行测试:

- $f = \cos, a = 0, b = \pi, n = 100$
- $f = \sin, a = 0, b = \frac{\pi}{2}, n = 100$
- $f = x^2, a = 0, b = 3, n = 100$

测试函数的容差设置为 10^{-3} 。

Notes

The End!

Notes

Notes

Notes

Notes
