Python基础语法精讲

嵩 天

函数定义与使用

嵩天

函数定义与使用

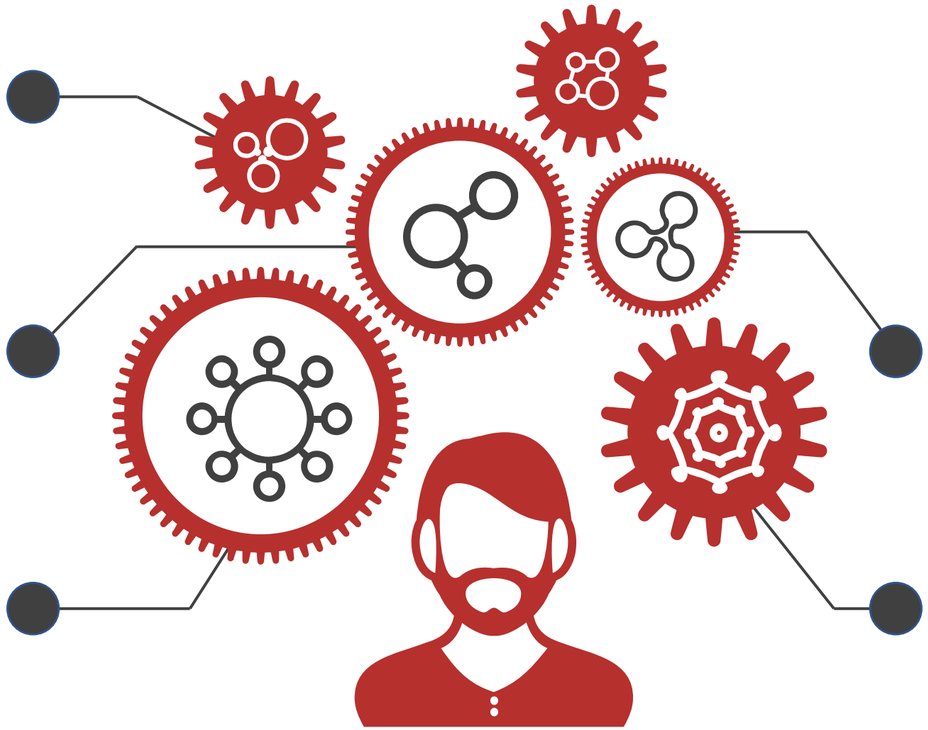
单元开篇

单元开篇

（1）函数的定义和调用

（2）函数的参数传递

（3）局部变量和全局变量



（4）lambda函数

（5）函数递归

函数定义与使用

单元开篇

函数相关的保留字

- def return global
- lambda

函数定义与使用

函数的定义和调用

Python123

函数的定义

函数是一段代码的抽象表示

```
def <函数名>(<参数(0个或多个)>) :
    <函数体>
    return <返回值(0个或多个)>
```

函数名　　参数

```
def fact(n) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
```

计算 n!

返回值

函数的定义

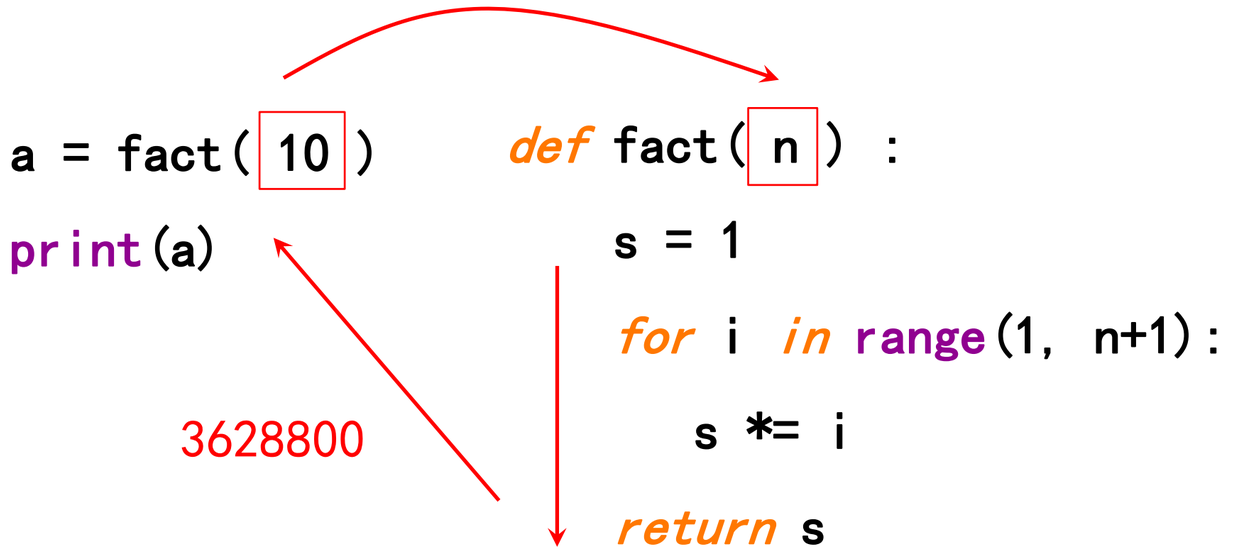
Python函数定义的注意事项

- 默认情况下，参数不需要类型声明
- 默认情况下，函数返回值不需要类型声明
- 参数和返回值可以为0个或多个

函数的调用

Python函数的定义和调用

```
def fact(n) :        #函数的定义
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
fact(10)             #函数的调用
```



函数定义与使用

函数的参数传递

Python 123

参数个数

函数可以有参数，也可以没有，但必须保留括号

```
def  <函数名>() :
        <函数体>
        return  <返回值>
```

```
def fact() :
    print("我也是函数")
```

参数传递

Python函数参数传递的4个方面

- 位置传递和名称传递
- 可选参数传递
- 可变参数传递：元组形式和字典形式
- 多返回值

位置传递和名称传递

函数调用时，参数可以按照位置或名称方式传递

```
def fact(n, m) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s//m
```

位置传递，按顺序

```
>>> fact( 10,5 )
725760
```

名称传递，按名称

```
>>> fact( m=5,n=10 )
725760
```

可选参数传递

可选参数：函数为某些参数指定默认值，构成可选参数

可选参数，放最后

```
def <函数名>(<非可选参数>, <可选参数>):
    <函数体>
    return <返回值>
```

可选参数传递

可选参数：函数为某些参数指定默认值，构成可选参数

计算 n!//m

```
def fact(n, m=1) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s//m
```

```
>>> fact(10)
3628800
>>> fact(10,5)
725760
```

可变参数传递

可变参数：函数可以接受不确定总数的参数变量

元组形式

```
def <函数名>(<参数>, *b ) :
    <函数体>
    return <返回值>
```

字典形式

```
def <函数名>(<参数>, **d ) :
    <函数体>
    return <返回值>
```

可变参数传递

可变参数：函数可以接受不确定总数的参数变量

```
def fact(n, *b):
    s = 1
    for i in range(1, n+1):
        s *= i
    for item in b:
        s *= item
    return s
```

元组形式

```
>>> fact(10,3)
10886400
>>> fact(10,3,5,8)
435456000
```

b = (3)

b = (3, 5, 8)

所有可变参数作为一个元组变量

可变参数传递

可变参数：函数可以接受不确定总数的参数变量

```
def fact(n, **d) :
    s = 1
    for i in range(1, n+1):
        s *= i
    for item in d:
        s *= d[item]
    return s
```

字典形式

```
>>> fact(10,a=3)
10886400
>>> fact(10,a=3,b=5,c=8)
435456000
```

d = {a:3}

d = {a:3, b:5, c:8}

所有名称传递方式的可变参数作为一个字典变量

函数的返回值

函数可以返回0个或多个结果

- return保留字用来传递返回值 或 表示函数结束
- 函数可以有返回值，也可以没有，可以有return，也可以没有
- return可以传递0个返回值，也可以传递任意多个返回值

函数的返回值

函数可以返回0个或多个结果

```
def fact(n, m=1) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s//m, n, m
```

```
>>> fact( 10,5 )
(725760, 10, 5)
>>> a,b,c = fact(10,5)
>>> print(a,b,c)
725760 10 5
```

元组类型

函数定义与使用

局部变量和全局变量

局部变量和全局变量

局部变量和全局变量

规则1: 局部变量和全局变量是不同变量

- 局部变量是函数内部的占位符，与全局变量可能重名但不同
- 函数运算结束后，局部变量被释放
- 使用*global*保留字在函数内部声明并使用全局变量

局部变量和全局变量

```
n, s = 10, 100
def fact(n) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
print(fact(n), s)
```

s是局部变量,与全局变量s不同

此处局部变量s是3628800

此处全局变量s是100

运行结果

```
>>>
3628800 100
```

局部变量和全局变量

```
n, s = 10, 100
def fact(n) :
    global s
    for i in range(1, n+1):
        s *= i
    return s
print(fact(n), s)
```

使用global保留字声明，此处s是全局变量s

此处s指全局变量s

此处全局变量s被函数修改

运行结果

```
>>>
362880000 362880000
```

局部变量和全局变量

规则2: 局部变量为组合数据类型且未创建，等同于全局变量

```
ls = ["F", "f"]
def func(a) :
    ls.append(a)
    return
func("C")
print(ls)
```

通过使用[]真实创建了一个全局变量列表ls

此处ls是列表类型，未真实创建则等同于全局变量

全局变量ls被修改

运行结果

```
>>>
['F', 'f', 'C']
```

局部变量和全局变量

```
ls = ["F", "f"]
def func(a) :
    ls = []
    ls.append(a)
    return
func("C")
print(ls)
```

通过使用[]真实创建了一个全局变量列表ls

通过使用[]真实创建了一个局部变量ls

局部变量ls被修改

全局变量ls未被修改

运行结果

```
>>>
['F', 'f']
```

局部变量和全局变量

使用规则总结

- 基本数据类型，无论是否重名，局部变量与全局变量不同
- 可以通过global保留字在函数内部声明全局变量
- 组合数据类型，如果局部变量未真实创建，则是全局变量

函数定义与使用

lambda函数

Python123

lambda函数

lambda函数返回函数名作为结果

- lambda函数是一种匿名函数，即没有名字的函数
- 使用lambda保留字定义，函数名是返回结果
- lambda函数用于定义简单的、能够在一行内表示的函数

lambda函数

<函数名> = *lambda* <参数>: <表达式>

等价于

```
def <函数名>(<参数>) :
    <函数体>
    return <返回值>
```

```
>>> f = lambda x, y : x + y
>>> f(10, 15)
25
>>> f = lambda : "lambda函数"
>>> print(f())
lambda函数
```

函数定义与使用

函数递归

递归的定义

函数定义中调用函数自身的方式

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & otherwise \end{cases}$$

- **链条：计算过程存在递归链条**
- **基例：存在一个或多个不需要再次递归的基例**

递归的实现

递归的实现：函数+分支结构

```
def fact(n):
    if n == 0 :
        return 1
    else :
        return n*fact(n-1)
```

- 递归本身是一个函数，需要函数定义方式描述
- 函数内部，采用分支语句对输入参数进行判断
- 基例和链条，分别编写对应代码

字符串反转

将字符串s反转后输出

- 函数 + 分支结构
- 递归链条
- 递归基例

```
def rvs(s):
    if s == "" :
        return s
    else :
        return rvs(s[1:])+s[0]
```

斐波那契数列

F(n) = F(n-1) + F(n-2)

$$F(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ F(n-1) + F(n-2) & otherwise \end{cases}$$

- 函数 + 分支结构
- 递归链条
- 递归基例

```
def f(n):
    if n == 1 or n == 2 :
        return 1
    else :
        return f(n-1) + f(n-2)
```

函数定义与使用

单元小结

单元小结

（1）函数的定义和调用

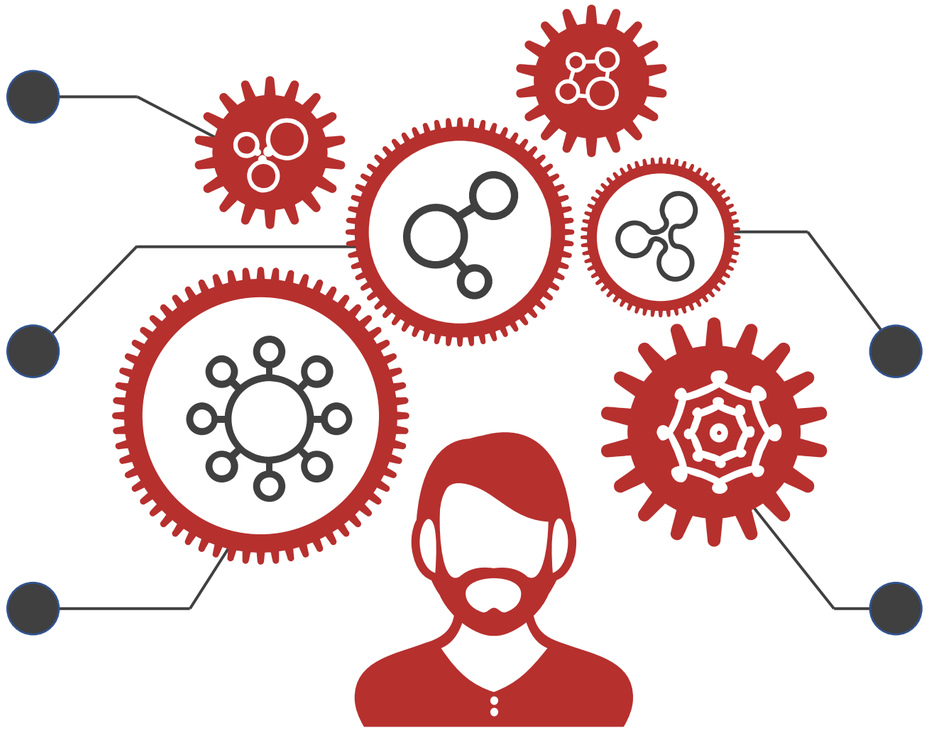
def定义、调用执行

（2）函数的参数传递

位置/名称、可变参数、可选参数

（3）局部变量和全局变量

global声明、组合数据类型引用



（4）lambda函数

基本定义和使用

（5）函数递归

函数+分支结构、递归链条、递归基例

函数定义与使用


Python 123
Thank you