

人工智能程序设计

M1 Python程序设计基础

4 函数

张 莉



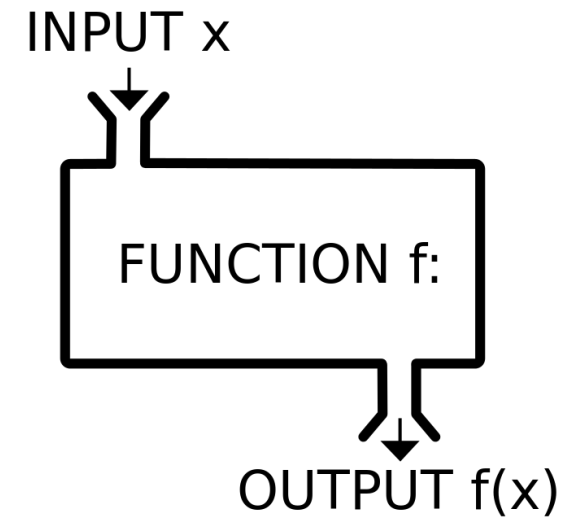
找前5个默尼森数

- P是素数且M也是素数，并且满足等式 $M=2^P-1$ ，则称M为默尼森数
- 例如 $P=5$ ， $M=2^P-1=31$ ，5和31都是素数，因此31是默尼森数。

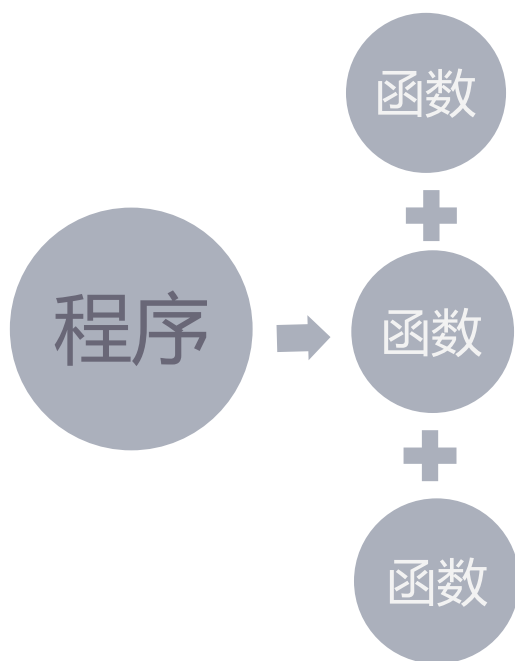
PRIME NUMBERS																									
2				3				5				7													
Any number under 100 which can not be divided by one of the above numbers is prime.																									
11				13				17				19													
Any number under 400 which can not be divided by one of the above numbers is prime.																									
23		29		31		37		41		43		47		53		59									
61		67		71		73		79		83		89		97											
Any number under 10,000 which can not be divided by one of the above numbers is prime.																									
101	103	107	109	113	127	131	137	139	143	151	157	163	167	173	179	181	187	191	193	197	199	211	223		
227	233	239	241	251	257	263	269	271	277	281	283	293	307	311	313	317	331	337	347	349	353	359			
373	379	383	389	397	409	419	421	431	433	439	443	449	457	461	463	467	473	479	487	491	499	503			
521	523	541	547	557	563	569	571	577	587	593	599	607	613	617	619	631	641	643	647	653	659				
673	677	683	691	701	709	727	731	733	737	743	751	757	761	769	773	787	797	809	811	823	827				
839	853	857	859	863	877	883	887	907	911	919	929	937	941	943	947	953	967	971	973	983	991	997			
Any number under 1,000,000 which can not be divided by one of the above numbers is prime.																									

函数

- 函数是一个独立的代码块
- 在解决大规模问题时采用“模块化”策略，将一个大而复杂的原始任务分解为多个较简单的子任务，再为每个简单的子任务设计算法
- 将描述其算法的一组语句封装为一个独立代码块，为每个独立代码块定义一个名字以及能与其他独立代码块通信的接口，这种独立的代码块定义就是函数



函数

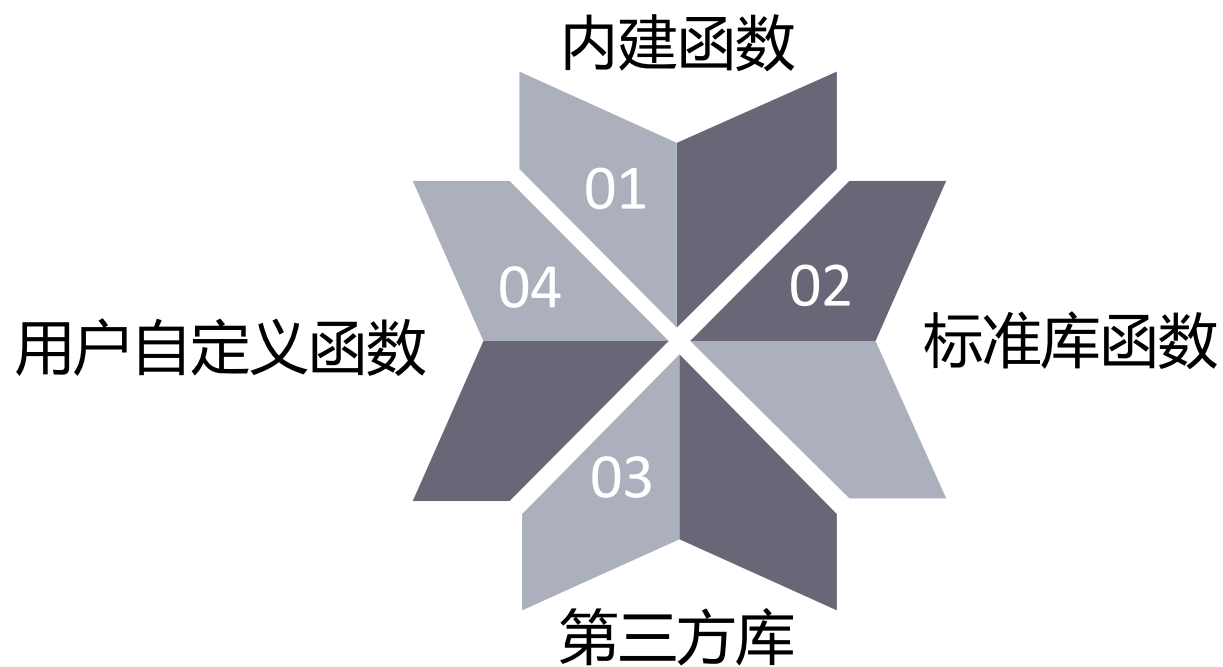


简化程序结构

降低程序开发和修改复杂度

提高程序可读性

Python中的函数

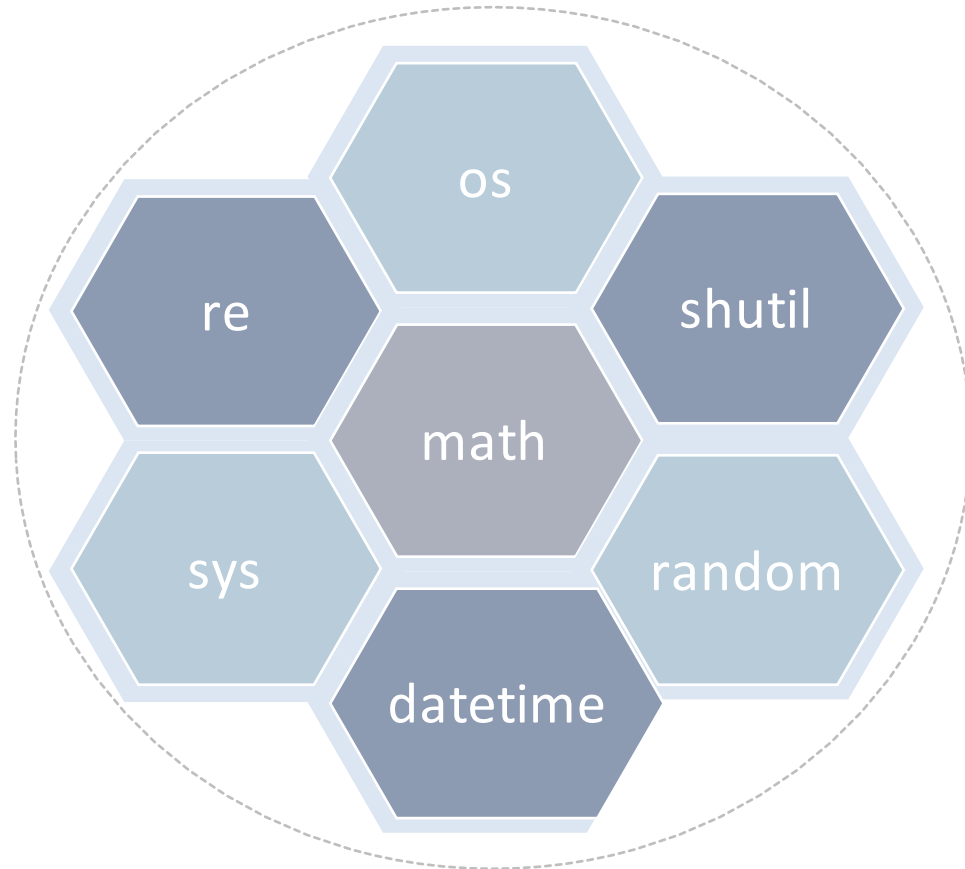


函数

1. 常用Python标准库函数
2. 自定义函数
3. 函数的参数
4. 递归函数
5. lambda函数与函数式编程
6. 变量作用域

1 人工智能程序设计 常用PYTHON标准库函数

常用Python 标准库函数



os模块中常用的处理文件及目录的函数

```
import os  
dir(os)
```

 Source

```
>>> import os  
>>> os.getcwd()  
'C:\\WINDOWS\\system32'  
>>> path = 'd:\\temp'  
>>> os.chdir(path)  
>>> os.getcwd()  
'd:\\temp'  
>>> os.rename('current.txt', 'new.txt')  
>>> os.remove('new.txt')  
>>> os.mkdir('d:\\temp\\tempdir')  
>>> os.rmdir('d:\\temp\\tempdir')
```

sys模块中标准输入和输出属性

```
import sys
dir(sys)
```

```
>>> for s in sys.stdin:
...     if s.strip() != '0':
...         print(s)
...     else:
...         break
```

```
>>> lst = []
>>> for item in sys.stdin:
...     lst.append(name)
```

```
>>> lst = []
>>> for item in sys.stdin:
...     name, score = item.split()
...     lst.append(name)
```

```
>>> sys.stdout.write('hello')
hello5
```

random模块中常用函数的功能和使用方法



```
>>> import random
>>> random.seed(100)
>>> random.random()
0.1456692551041303
>>> random.random()
0.38859914082194214
>>> random.choice(['C++', 'Java', 'Python'])
'Java'
>>> random.randint(1, 100)
37
>>> random.randrange(0, 10, 2)
4
>>> random.uniform(5, 10)
5.776718084305783
```

```
import random
dir(random)
```


random模块中常用函数的功能和使用方法

Source

```
>>> import random
>>> random.sample(range(100), 10)
[16, 49, 26, 6, 61, 64, 29, 28, 34, 72]
>>> nums = [1002, 1004, 1001, 1005, 1008]
>>> random.shuffle(nums)
>>> nums
[1002, 1008, 1001, 1005, 1004]
```



datetime模块中的函数



```
>>> from datetime import date
>>> date.today()
datetime.date(2019, 3, 15)
>>> import datetime
>>> tm = datetime.time(23, 20, 35)
>>> from datetime import time
>>> time.isoformat(tm)
'23:20:35'
```

datetime模块中的函数



```
>>> from datetime import datetime
>>> dt = datetime.now()
>>> dt
datetime.datetime(2019, 3, 15, 23, 25, 4, 125366)
>>> print(dt.strftime('%a, %b %d %Y %H:%M'))
Fri, Mar 15 2019 23:25
```

形式1	形式2	含义
%a	%A	星期
%b	%B	本地月份
%d		日期
%y	%Y	年份
%H	%I	小时数
%M		分钟数

timestamp()和fromtimestamp()



```
>>> dt = datetime(2018, 4, 9, 23, 29)
>>> print(dt)
2018-04-09 23:29:00
>>> ts = dt.timestamp()
>>> ts
1523287740.0
>>> print(datetime.fromtimestamp(ts))
2018-04-09 23:29:00
```

2 人工智能程序设计 自定义函数

自定义函数的创建



```
>>> def printStr(x):  
    '''print the string'''  
    print(x)
```

def 函数名([参数表]):
 [''文档字符串'']
 函数体

```
>>> from f_name import printStr  
>>> print(printStr.__doc__)
```

函数的返回



```
>>> def square(x, y):  
    '''  
    计算参数的和与差  
    '''  
    return x + y, x - y
```

return 表达式1, 表达式2, ..., 表达式n

函数的调用

```
>>> printStr('Hi, Python!')
```

```
Hi, Python!
```

```
>>> x, y = square(3, 5)
```

```
>>> x
```

```
8
```

```
>>> y
```

```
-2
```

```
>>> from pStr import printStr
```

```
>>> printStr('Hi, Python!')
```

```
Hi, Python!
```




pStr.py

例 求1~100之间的所有素数

- 输出1-100之间的素数

Output:

2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67
71 73 79 83 89 97

 *# Filename: prime.py*

```
from math import sqrt
def isprime(x):
    if x == 1:
        return False
    k = int(sqrt(x))
    for j in range(2, k+1):
        if x % j == 0:
            return False
    return True
for i in range(2, 101):
    if isprime(i):
        print(i, end = ' ')
```

例 求1~100之间的所有素数

File

```
# Filename: prime.py
from math import sqrt
def isprime(x):
    if x == 1:
        return False
    k = int(sqrt(x))
    for j in range(2, k+1):
        if x%j == 0:
            return False
    return True
for i in range(2,101):
    if isprime(i):
        print(i, end = ' ')
```

File

```
if __name__ == "__main__":
    for i in range(2, 101):
        if isprime(i):
            print(i, end = ' ')
```

例 字符串循环移动

- 自定义函数move_substr(s, flag, n)，将传入的字符串s按照flag（1代表循环左移，2代表循环右移）的要求左移或右移n位，结果返回移动后的字符串，若n超过字符串长度则结果返回-1。
- __main__模块中从键盘输入字符串、左移和右移标记以及移动的位数，调用move_substr()函数若移动位数合理则将移动后的字符串输出，否则输出“the n is too large”。

例 字 符 串 循 环 移 动

```
def moveSubstr(s, flag, n):  
    if n > len(s):  
        return -1  
    else:  
        if flag == 1:  
            return s[n:] + s[:n]  
        else:  
            return s[-n:] + s[:-n]  
  
if __name__ == "__main__":  
    s, flag, n = input("enter the 'string,flag,n': ").split(',')  
    result = moveSubstr(s, int(flag), int(n))  
    if result != -1:  
        print(result)  
    else:  
        print('the n is too large')
```

例 模拟一个简易的用户注册和登录系统



Filename: account.py

`account = {'Zhangsan': '123456'}` *# account为全局变量*

`def sign_up():`

`user_name = input("Please input your user name: ")`

`while user_name in account.keys():`

`user_name = input("User name exists, please choose another one:")`

`password = input("Please input your password: ")`

`account[user_name] = password`

`print("Successfully sign up!")`

例 模拟一个简易的用户注册和登录系统



```
def sign_in():
    user_name = input("Please input your user name: ")
    if user_name not in account.keys():
        print("User name not found.")
    else:
        count = 0
        password = input("Please input your password: ")
        while account[user_name] != password:
            count += 1
            if count >= 3 :
                print("Bye - bye")
                break
            password = input("Wrong password, please input again: ")
        if account[user_name] == password:
            print("Login success!")
```

例 模拟一个简易的用户注册和登录系统



```
if __name__ == '__main__':  
    while True:  
        cmd = input("Sign Up or Sign In? Please input 0 or 1:")  
        while cmd != '0' and cmd != '1':  
            print('Wrong command, please input again: ')  
            cmd = input("Sign Up: 0, Sign in: 1")  
        if cmd == '0':  
            sign_up()  
            continue  
        if cmd == '1':  
            sign_in()  
            break
```

例 模拟一个简易的用户注册和登录系统

Output:

Sign Up or Sign In? Please input 0 or 1:

0

Please input your user name: Lisi

Please input your password: 123456

Successfully sign up!

Sign Up or Sign In? Please input 0 or 1:

1

Please input your user name: Lisi

Please input your password: 123456

Login success!

3 人工智能程序设计 函数的参数

1. 位置参数

Source

```
>>> def printGrade(name, stuID, grade):  
    print("{0}({1})'s grade is {2}.".format(name, stuID, grade))  
>>> printGrade('Mary', '1002', 'A')  
Mary(1002)'s grade is A.
```

Source

```
>>> printGrade('A', '1002', 'Mary')  
A(1002)'s grade is Mary.
```

2. 关键字参数

Source

```
>>> def printGrade(name, stuID, grade):  
    print("{0}({1})'s grade is {2}.".format(name, stuID, grade))  
>>> printGrade(name = 'Jerry', stuID = '1005', grade = 'B')  
Jerry(1005)'s grade is B.
```

让调用者通过使用参数名区分参数
允许改变参数列表中的参数顺序
调用时每个参数的含义更清晰

2. 关键字参数

Source

```
>>> def f(x, y):  
    '''x and y both correct words or not'''  
    if y:  
        print(x, 'and y both correct')  
    print(x, 'is OK')
```

```
>>> f(68, False)
```

```
68 is OK
```

```
>>> f(y = False, x = 68)
```

```
68 is OK
```


```
>>> f(y = False, 68)
```


```
SyntaxError: non-keyword arg after keyword arg
```


```
>>> f(68, y = False)
```


```
68 is OK
```

3. 默认参数


>>> `def area(r, pi = 3.14159):`
 `return pi * r * r`


>>> `area(3)`
`28.274309999999996`


>>> `area(4, 3.14)`
`50.24`


>>> `area(pi = 3.14, r = 4)`
`50.24`

3. 默认参数

S_{ource}

```
>>> def printGrade(name, className = 'Courage', grade):  
    print("{0}({1})'s grade is {2}.".format(name, className, grade))  
>>> printGrade('Mary', 'A')
```

SyntaxError: non-default argument follows default argument

3. 默认参数



```
>>> def printGrade(name, grade, className = 'Courage'):
    print("{0}({1})'s grade is {2}.".format(name, className, grade))
>>> printGrade('Mary', 'A')
Mary(Courage)'s grade is A.
```

4. 可变长参数—可变长位置参数

- 允许传递一组数据给一个形参，形参前 “*” 号是可变长位置参数的标记，用来收集其余的位置参数，将它们放到一个元组中



```
>>> def greeting(args1, *tupleArgs):  
    print(args1)  
    print(tupleArgs)  
>>> greeting('Hello,', 'Wangdachuan', 'Liuyun', 'Linling')  
Hello,  
(('Wangdachuan', 'Liuyun', 'Linling'))
```


4. 可变长参数—可变长位置参数



```
>>> def greeting(args1, *tupleArgs):  
    print(args1)  
    print(tupleArgs)  
>>> names = ('Wangdachuan', 'Liuyun', 'Linling')  
>>> greeting('Hello,', *names)  
Hello,  
('Wangdachuan', 'Liuyun', 'Linling')
```

4. 可变长参数—可变长关键字参数

- 用两个星号标记可变长的关键字参数。可变长关键字参数允许传入多个(可以是0个)含参数名的参数,这些参数在函数内自动组装成一个字典。

 Source

```
>>> def assignment(**dictArgs):  
        print(dictArgs)  
>>> assignment(x = 1, y = 2, z = 3)  
{'x': 1, 'z': 3, 'y': 2}  
>>> data = {'x': 1, 'z': 3, 'y': 2}  
>>> assignment(**data)  
{'x': 1, 'z': 3, 'y': 2}
```

4. 可变长参数—可变长位置参数 和可变长关键字参数

S
ource

```
>>> def greeting(args1, *tupleArgs, **dictArgs):  
    print(args1)  
    print(tupleArgs)  
    print(dictArgs)  
>>> names = ['Wangdachuan', 'Liuyun', 'Linling']  
>>> info = {'schoolName': 'NJU', 'City': 'Nanjing'}  
>>> greeting('Hello,', *names, **info)  
Hello,  
( 'Wangdachuan', 'Liuyun', 'Linling' )  
{ 'City': 'Nanjing', 'schoolName': 'NJU' }
```

例 实现用户信息注册登记

- 要求必须登记姓名，性别和手机号码，其他如年龄、职业等信息不强制登记。



Filename: register.py

```
def register(name, gender, phonenumber, **otherinfo):  
    ''' register users information '''  
    print('name: ', name, 'gender: ', gender, 'phone num: ', phonenumber)  
    print('other information: ', otherinfo)
```



例 实现用户信息注册登记



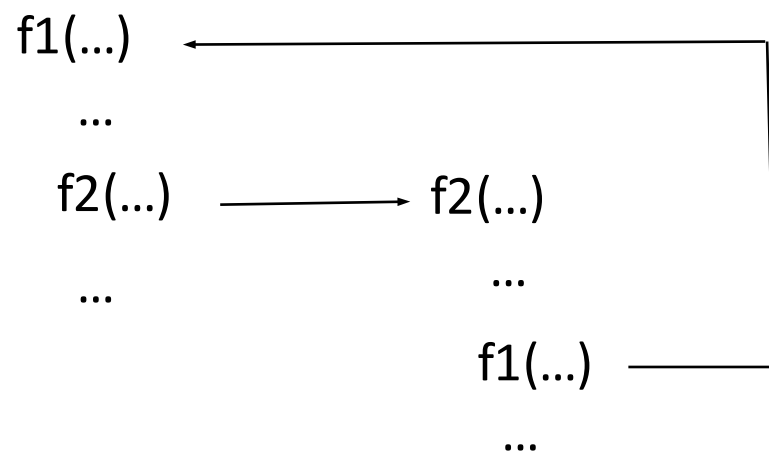
```
>>> register('Chenqian', 'M', '11111111111')  
name: Chenqian gender: M phone num:  
11111111111  
other information: {}
```



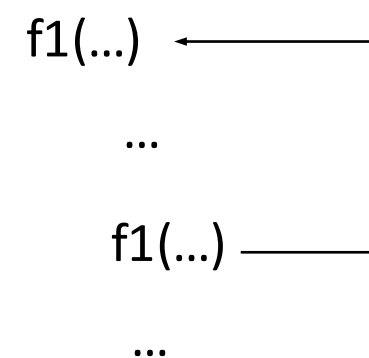
```
>>> otherinfo = {'age': 24, 'city': 'Nanjing', 'job': 'teacher'}  
>>> register('Limei', 'F', '22222222222', **otherinfo)  
name: Limei gender: F phone num: 2222222222  
other information: {'age': 24, 'city': 'Nanjing', 'job': 'teacher'}
```


4 人工智能程序设计 递归函数

直接递归和间接递归



(a) 间接递归调用




(b) 直接递归调用

递归是特殊的嵌套调用，是对函数自身的调用

正确的递归调用的要求

- 有一个比原始调用规模小的函数副本
- 有基本情况即递归终止条件



```
def f():  
    f()
```

无穷递归 (infinite recursion)

递归调用的过程

- 每一次递归调用要解决的问题都要比上一次的调用简单，规模较大的问题可以往下分解为若干个规模较小的问题，规模越来越小最终达到最小规模的递归终止条件（基本情况）
- 解决完基本情况后函数沿着调用顺序逐级返回上次调用，直到函数的原始调用处结束
- 一般会包含一个选择结构，条件为真时计算基本情况并结束递归调用，条件为假时简化问题执行副本继续递归调用。

例 编写递归函数计算n的阶乘

$$n! = \begin{cases} 1 & (\text{当 } n=1) \\ n \times (n-1)! & (\text{当 } n>1) \end{cases}$$

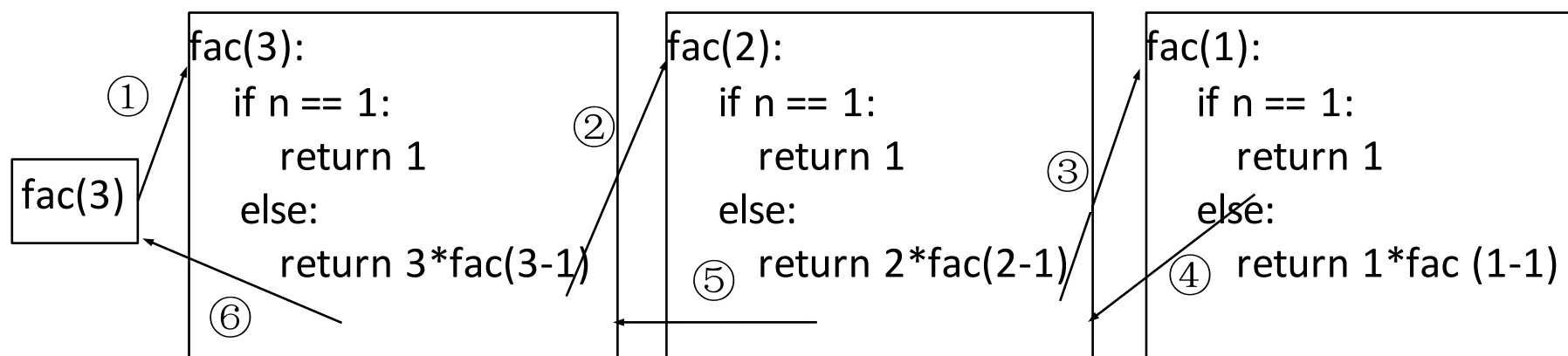
n的阶乘的定义是一种递归定义



Filename: fac.py

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

例 编写递归函数计算n的阶乘



递归

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

```
s = input("input a string: ")  
print(proc(s))
```

输入step

递归深度的设定

- 查看递归深度

```
>>> import sys
```

```
>>> sys.getrecursionlimit()
```

```
1000
```

- 手工修改默认值

```
sys.setrecursionlimit(2000)
```


5 人工智能程序设计 LAMBDA函数和函数式编程

Lambda函数

lambda函数又称为匿名函数，即没有具体的函数名

lambda函数的目的是让用户快速地定义单行函数，简化用户使用函数的过程。

```
def my_add(x, y) : return x + y
```


```
lambda x, y : x + y
```


```
my_add = lambda x, y : x + y
```

```
>>> my_add(3, 5)
```

```
8
```

lambda函数

 `>>> def addMe2Me(x):`
 'apply operation + to argument'
 `return x + x`
`>>> addMe2Me(5)`
`10`

 `>>> r = lambda x : x + x`
`>>> r(5)`
`10`

例 编写函数计算平均成绩—lambda函数

Source

```
>>> dScores = {'Jerry': [87, 85, 91], 'Mary': [76, 83, 88], 'Tim':  
[97, 95, 89], 'John': [77, 83, 81]}
```

```
>>> a = sorted(dScores.items(), key = lambda d:d[0])  
[('Jerry', [87, 85, 91]), ('John', [77, 83, 81]), ('Mary', [76, 83,  
88]), ('Tim', [97, 95, 89])]
```

```
>>> a = sorted(dScores.items(), key = lambda d:d[1][0])  
[('Mary', [76, 83, 88]), ('John', [77, 83, 81]), ('Jerry', [87, 85,  
91]), ('Tim', [97, 95, 89])]
```

例 编写函数计算平均成绩——lambda函数



```
def search(scores):  
    t = sorted(scores.items(), key = lambda d : (d[1][0] +  
        d[1][1] + d[1][2]) // 3)  
    return t[len(t)-1][0], t[0][0]
```

函数式编程

- 函数式编程的主要由3个基本函数和1个算子构成

基本函数：

map()、reduce()、
filter()

算子(operator)：
lambda

```
>>> lst = [3, 2, 5, 8, 1]
>>> list(map(lambda x: x*2, lst))
[6, 4, 10, 16, 2]
```

```
>>> lst = [1, 2, 3, 4, 5, 6]
>>> list(filter(lambda x: x%2 == 0, lst))
[2, 4, 6]
```

```
>>> from functools import reduce
>>> lst = [1, 2, 3, 4, 5]
>>> reduce(lambda x, y: x + y, lst)
15
```

6 变量的作用域

人工智能程序设计

变量作用域



```
>>> def f(): x = 5
```

```
>>> f()
```

```
>>> print(x)
```

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

print(x)

NameError: name 'x' is not defined

变量作用域



当在函数中使用未确定的变量名时，搜索变量名的顺序遵循LEGB法制

作用域

- 作用域会生成一个命名空间
(namespace)
- 命名空间是从名称 (标识符)
到对象的映射
- 不同命名空间中的名字之间没
有关系



对不同作用域同名变量的处理

 Source


```
>>> def f(): x = 5
>>> x = 3
>>> f()
>>> print(x)
3
```

 Source

```
>>> def f(): y = 5
>>> x = 3
>>> f()
>>> print(x)
3
```


函数内部使用全局变量

慎用全局变量

 `>>> def f():`
 `y = 5`
 `x = 8`
 `print(x + y)`
`>>> x = 3`
`>>> f()`
`13`


局部变量和全局变量同名时

在局部变量（包括形参）和全局变量同名时，局部变量屏蔽（Shadowing）全局变量



```
>>> x = 3
>>> def f():
        x = 5
        print(x ** 2)
>>> f()
```

函数内部同时出现同名局部变量和全局变量

 Source

```
>>> x = 3
>>> def f():
    print(x ** 2)
    x = 5
    print(x ** 2)
>>> f()
```

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>


f()

File "<pyshell#2>", line 2, in f

print(x ** 2)

UnboundLocalError: local variable 'x' referenced before assignment

函数内部同时出现局部变量和全局变量

 Source

```
>>> x = 3
>>> def f():
    global x
    print(x ** 2)
    x = 5
    print(x ** 2)
>>> x = 3
>>> f()
9
25
```

使用关键字 `global` 声明将使用全局变量

M1.4 小结

- 01 常用Python标准库函数
- 02 自定义函数
- 03 函数的参数
- 04 递归函数
- 05 lambda函数与函数式编程
- 06 变量作用域