小伙伴计划暑期学习营——零基础Python入门

第二讲：Python基础语法
——基本语法结构

张智帅 电子系

清华大学学生学业与发展指导中心

2019-2020学年夏季学期

第二讲：基本语法结构

- 数据结构
  - 字符串
  - 列表
  - 字典
  - 元组
  - 集合
- 控制结构

数据结构-列表

- 列表（list）：Python的主力
- **索引**（indexing）：**通过编号访问元素**

```
b_list = ["h", "u", "a", 110]
b_list[0]
b_list[1]
b_list[-1]
```

从零开始

负数=倒数

- **切片**（slicing）：访问**特定范围内的元素**

```
b_list = ["h", "u", "a", 110]
print(b_list[1:2])  # ['u']
print(b_list[:3])  # ['h', 'u', 'a']
print(b_list[-1:])  # [110]
```

前闭后开

数据结构-列表

- 通过编号访问元素
  - **从零开始**
  - **负数=倒数**

- **谨记：不要越界**

数据结构-列表操作 回 囘 囲 圓

- 列表是**可变**的。基本的操作：**索引、切片、增删查改、排序**、清空、弹出、复制、计数、相加、相乘、成员资格检查、获取序列属性（长度、最大值、最小值）……**根据需要，现用现查**
  - 修改：元素赋值、切片赋值

```
b = ["THU", "PKU", 1, 2, [886, 233]]

# 元素赋值
b[2] = "THU"  # 修改列表 b 的第 2 号元素
print(b)

# 切片赋值
b[:2] = ["shiyida", "shierda"]  # 修改列表 b 的前两个元素的切片
print(b)
```

数据结构-列表操作

- 增删查

```
b.append("Good")  # 在列表 b 的最后增加一个元素
print(b)
b.insert(2, "Bad")  # 在位置 2 处插入 "Bad" 这个元素
print(b)
del b[1]  # 根据编号删除，删除列表 b 的第 1 号元素
print(b)
b.remove("Bad")  # 根据元素删除，删除列表中 "Bad" 这个元素
print(b)
i = b.index("THU")  # 查找指定值第一次出现的索引
print(i)
print(b[i])
```

- 排序

```
# 排序，需保证列表内的元素能够比大小
x = [4, 5, 1, 2, 3, 7, 6]
x.sort()  # 正序排序
print(x)

x.sort(reverse=True)  # 倒序排列
print(x)
```

数据结构-元组

- 列表和元组的主要不同：
  - 列表用方括号，元组用圆括号
  - 列表可变，元组不可变
  - 元组不支持元素赋值、切片复制，也不支持append、insert等修改方法

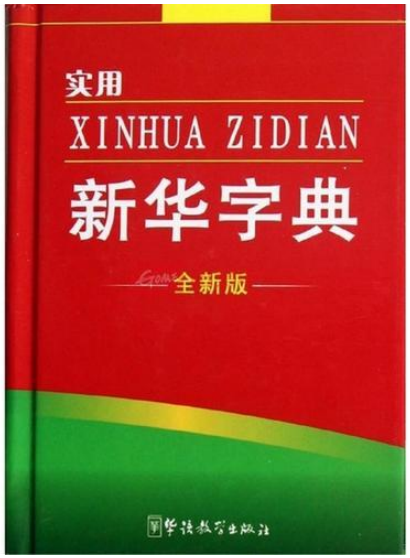
```
t = [1, 2, 3]  # 列表
t[2] = 4
print(t)

t = (1, 2, 3)  # 元组
t[2] = 4
print(t)
```

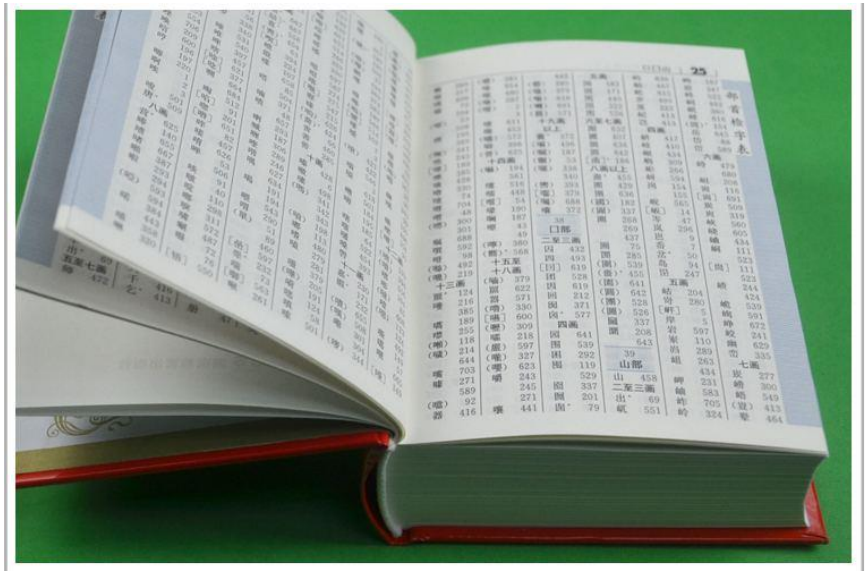
```
[1, 2, 4]
Traceback (most recent call last):
  File "e:\Testfield\summer_python\Lecture_2\L2_tuple.py", line 7, in <module>
    t[2] = 4
TypeError: 'tuple' object does not support item assignment
```

- 更安全：出于某种考虑，需要禁止修改序列

数据结构-字典

- 内置**映射**类型：空间换时间，读写速度快





- 字典（dictionary）：通过键（key）访问值（value）

①大括号　　③key与value之间：英文冒号

```
names = {"Alex": 2020010483, "Jack": 2019880789, "Tim": [2015050888,"Phd"]}
```

②键值对之间，英文逗号

- 用途：键易得，值比较复杂的数据；需要高速查找的数据

数据结构-字典

- 元素访问方式：**只能通过key访问value**；不能通过value访问key；不能通过编号访问元素。

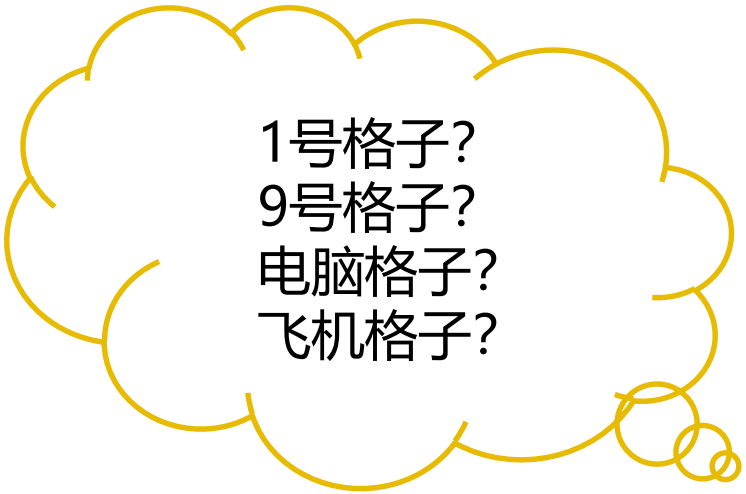
```
names = {"Alex": 2020010483, "Jack": 2019880789, "Tim": [2015050888,"Phd"]}

print(names["Alex"])
print(names["Tim"])
print(names[1])
```

```
2020010483
[2015050888, 'Phd']
Traceback (most recent call last):
  File "e:\Testfield\summer_python\Lecture_2\L2_dict_2.py", line 7, in <module>
    print(names[1])
KeyError: 1
```

- key必须是不可变对象，而且不可以重复；value没有限制

数据结构-字典

- **只能通过key访问value**
  - 不能通过value访问key；
  - 不能通过编号访问；



- **谨记：不要访问不存在的键**

数据结构-字典

回 囘 囲 圓

- 列表是**可变**的。基本的操作：**增删查改**、清空、弹出、复制、更新、访问可能不存在的键、获取键列表、获取值列表、获取键值对列表……**根据需要，现用现查**

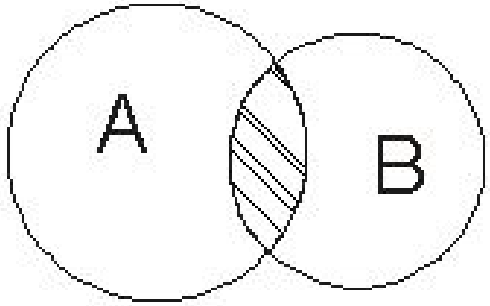
```
# 常用字典操作：增删查改
names["Eva"] = [2016050999, "Master"]  # 将新的值关联到键 "Eva" 上（新增键值对）
print(names)
print(names["Eva"])  # 现在可以访问键 "Eva" 了

del names["Jack"]  # 删除键为 "Jack" 的键值对
print(names)

search = "Alex" in names  # 检查字典 names 是否包含键为 "Alex" 的键
print(search)
print("Jimmy" in names)  # 检查字典 names 是否包含键为 "Jimmy" 的键

names["Eva"] = [2020010001, "Professor"]  # 将新的值关联到键 "Eva" 上（覆盖原来的值）
print(names)
```

数据结构-集合

回 囘 囲 圓

- 集合（set）：**无序、元素不重复。**

```
aa = {"瓜农", "套娃", "渔夫"} 大括号
print(aa[0])  # 集合不支持编号索引
```

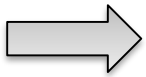


Venn diagram

- 用途：可用于关系运算（交并补），例如列表去重。

```
# 列表去重
croud = ["瓜农", "套娃", "渔夫", "瓜农", "套娃", "渔夫", "瓜农", "套娃", "渔夫", "瓜农"]
s = set(croud)  # 把列表强制类型转换为集合
print(s)
```

```
(base) E:\Testfield\summe
{'瓜农', '渔夫', '套娃'}
```



```
(base) E:\Testfield\summe
{'渔夫', '套娃', '瓜农'}

(base) E:\Testfield\summe
{'套娃', '渔夫', '瓜农'}
```

- 其他操作，**根据需要，现用现查**

数据结构-字符串

- 没有“字符”与“字符串”的差别；单引号与双引号基本无区别
- 字符串（str）与列表一样，可以索引、切片……但是字符串**不可变**，不支持元素赋值、切片赋值等直接修改。

```
s = ["T", "s", "i", "n", "g", "h", "u", "a"]
print(s[0])  # 索引
print(s[1:4])  # 切片
s[0] = "t"  # 索引赋值
print(s)

s = "Tsinghua"
print(s[0])  # 索引
print(s[1:4])  # 切片
s[0] = "t"  # 索引赋值
print(s)
```

```
T
['s', 'i', 'n']
['t', 's', 'i', 'n', 'g', 'h', 'u', 'a']
T
sin
Traceback (most recent call last):
  File "e:\Testfield\summer_python\Lecture_2\L2_string.py"
    s[0] = "t"
TypeError: 'str' object does not support item assignment
```

数据结构-字符串

回 囘 囲 圓

- 字符串操作：**拼接、替换、拆分、合并**、改变大小写、两边填充字符、删除两边空白、判断属性（是否只有数字、字母、空格，是否大写、小写、首字母大写）……**根据需要，现用现查**

```
a = "Tsing"
b = "hua"
s = a + b  # 字符串拼接
print(s)

s2 = s.replace("Ts", "tSssss")  # 把字符串 s 中的 "Ts" 替换为 "tSssss"
print(s2)
```

```
Tsinghua
tSssssinghua
```

```
date = "2020-7-24-3-44"
lst = date.split("-")  # 以 "-" 为分隔符把字符串 date 拆分为列表 s
print(lst)

inter = " + "
s2 = inter.join(lst)  # 以字符串 inter 为分隔符把列表 s 拼成字符串 s2
print(s2)
```

```
['2020', '7', '24', '3', '44']
2020 + 7 + 24 + 3 + 44
```

数据结构-字符串格式化

- 字符串格式化：本质上并不是字符串常量，而是一个**在运行时求值**的表达式

- 可以控制输出的格式，数据类型、左右对齐、宽度、小数位数.....
  - **不需要记，需要用的时候现查。**

Python 计算三角形的面积

Python3 实例

以下实例为通过用户输入三角形三边长度，并计算三角形的面积：

实例(Python 3.0+)

```
# -*- coding: UTF-8 -*-

# Filename : test.py
# author by : www.runoob.com


a = float(input('输入三角形第一边长: '))
b = float(input('输入三角形第二边长: '))
c = float(input('输入三角形第三边长: '))

# 计算半周长
s = (a + b + c) / 2

# 计算面积
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print('三角形面积为 %0.2f' %area)
```

占位符

浮点数，保留2位小数

数据结构-字符串格式化

- 百分号方式、format方式、f-string

```
GPA = 4.0
name = "Jack"

info = "Jack's GPA is 4.0, but he still isn't happy"  # 纯字符串
info1 = "%s's GPA is %0.1f, but he still isn't happy " % (name, GPA)  # 百分号格式化
info2 = "{}'s GPA is {}, but he still isn't happy".format(name, GPA)  # format 格式化
info3 = f"{name}'s GPA is {GPA}, but he still isn't happy"  # f-string 格式化
```

```
Jack's GPA is 4.0, but he still isn't happy
Jack's GPA is 4.0, but he still isn't happy
Jack's GPA is 4.0, but he still isn't happy
Jack's GPA is 4.0, but he still isn't happy
```

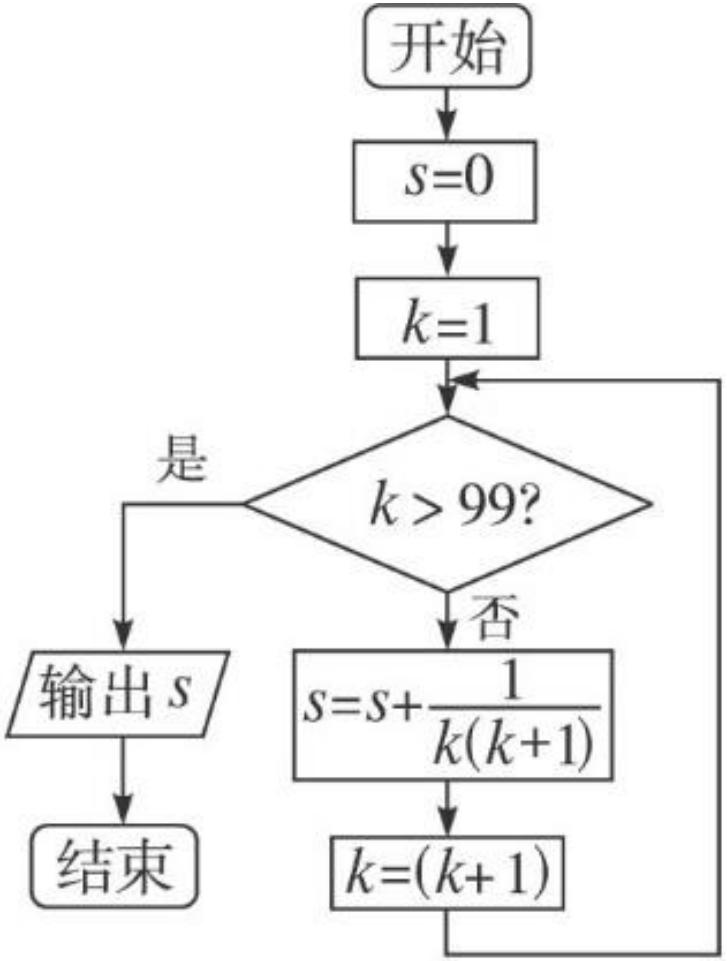
- Python字符串格式化-学这些就够用了

数据结构-回顾


通过编号访问元素
序列
字符串
"Tsinghua"
不可修改
列表
["THU", "PKU", 1, 2, [886, 233]]
元组
(1, 2, 3)
不可修改
通过键访问值
映射
字典
{"Alex": 2020010483, "Jack": 2019880789}
既不是序列
也不是映射
集合
{"abc", "dfg", "mnk"}

第二讲：基本语法结构

- 数据结构
- 控制结构
  - 条件分支
  - 循环
  - 函数



控制结构-条件分支

- 逻辑值：
  - 真=1=True=成立
  - 假=0=False=不成立

```
print(61.5 >= 60)  # True
print(61.5 == 60)  # False
print(61.5 <= 60)  # False
```

- 单路条件分支：if-else语句

```
score = 61.5
if score >= 60:
    grade = "P"
else:
    grade = "F"
print(grade)
```

英文冒号

缩进块

注意缩进层次

```
score = 61.5
if score >= 60:
    grade = "P"
else:
    grade = "F"
    print(grade)
```

无输出！

控制结构-条件分支

- 多路条件分支：if-elif-else语句

```
if score == 100:
    grade = "A+"
elif score >= 95:
    grade = "A"
else:
    grade = "A-"
print(grade)
```

- 嵌套条件分支：注意**缩进的匹配**

```
22 if score >=90:
23     if score == 100:
24         grade = "A+"
25     elif score >= 95:
26         grade = "A"
27     else:
28         grade = "A-"
29 else:
30     if score >= 60:
31         grade = "P"
32     else:
33         grade = "F"
34         print(grade)
35     print(grade)
36 print(grade)
```

三者作用不同

控制结构-循环

- 循环（loop）：重复执行代码

- for循环：**for 变量 in 可迭代对象:**

  - 把可迭代对象（str、list、tuple、range……）中的每个元素代入变量x，然后执行缩进块的语句；

```
x_sequence = [1, 2, 3, 4, 5, 6, 7, 8]
for x in x_sequence: 英文冒号
    print(x)   缩进块
```

  - 确定性**遍历**，迭代次数已知

控制结构-循环

- 范围（range）：可迭代对象，创建整数序列

```python
# 从 1 累加到 100，需要手写一个这么长的列表吗？直接生成整数序列
for i in range(3):  # 生成一个范围为 [0,3) 的整数序列
    print(i)

for i in range(2, 10):  # 生成一个范围为 [2,10) 的整数序列
    print(i)

for i in range(0, 100, 9):  # 生成一个范围为 [0,100)，每 9 个数取一个数的整数序列
    print(i)
```

控制结构-循环

```
x = 1
while x < 5:
    print(x)
    x += 1
```

```
x = 1
while x < 5:
    print(x)
```

- while循环：根据条件进行循环
  - 迭代次数不确定，可能出现死循环
  - 只要能够使用for循环，就不要使用while循环。

需要强制停止程序

- 中止循环
  - 跳出**整个**循环：break
  - 跳出**当前**循环：continue

```
age = 1
while age < 100:
    age += 1
    if age == 35:
        break
    print(age)
```

```
age = 1
while age < 100:
    age += 1
    if age == 35:
        continue
    print(age)
```

- 其他循环操作：带索引的for循环（enumerate)、遍历字典（按照键、值、键值对）……**根据需要，现用现查**

控制结构-循环-列表生成式

- 列表生成式（List Comprehensions）：[返回值 for... in... if...]

- 用for循环遍历列表里面的元素，生成操作后的新列表

```
# # 列表生成式
lst = [1, 2, 3, 4, 5, 6, 7, 8]
numbers = [x + 2 for x in lst]
print(numbers)  # [1, 2, 3, 4, 5, 6, 7, 8]

numbers = [x * x for x in range(10)]
print(numbers)  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# # 列表生成式 + 条件判断
numbers = [x for x in range(0, 100, 9) if x % 2 == 0]
print(numbers)  # [0, 18, 36, 54, 72, 90]
```

- 用途：方便、简洁地生成列表

函数

- 函数是具有特定功能的**可重用**代码片段
  - 是最基本的代码抽象的方式

- 定义函数

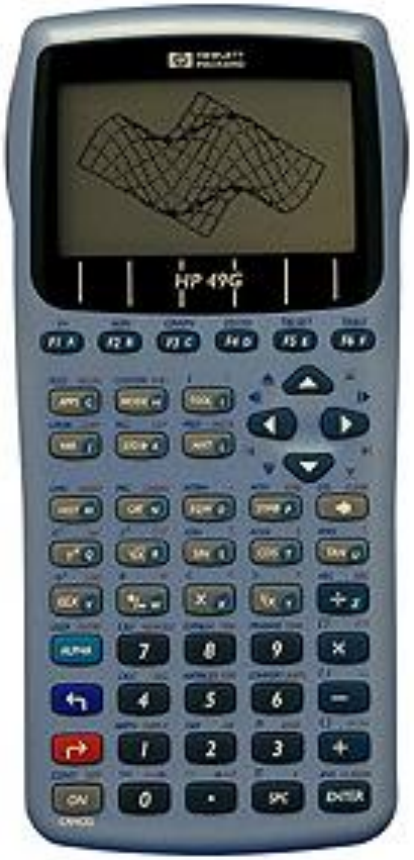
关键字　函数名　参数　冒号

```
def judge_PF(score):
    if score >= 60:
        grade = "P"
    else:
        grade = "F"
    return grade
```

函数体

返回值

内部细节



外部接口

函数

- 调用函数：函数通过参数接收所需要的信息
  - 位置参数、关键字参数、默认参数
  - https://www.liaoxuefeng.com/wiki/1016959663602400/1017261630425888

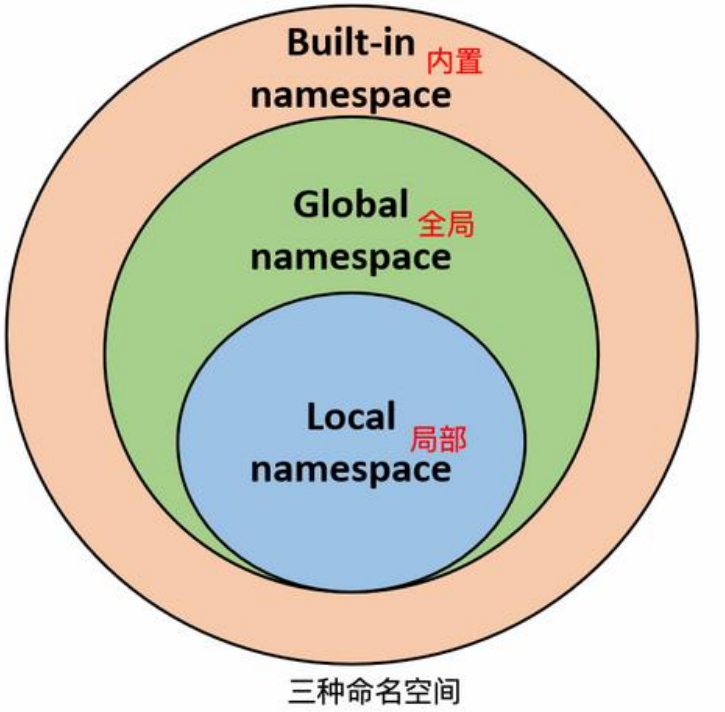
```
def judge_PF(score):
    if score >= 60:
        grade = "P"
    else:
        grade = "F"
    return grade


print(judge_PF(78))
print(judge_PF(score=58))
print(judge_PF(score=64))
```

定义函数

调用函数

函数

- 命名空间（作用域）



三种命名空间

```
score = 50


def judge_PF(score):
    if score >= 60:
        grade = "P"
    else:
        grade = "F"
    return grade


print(judge_PF(78))
```

局部作用域>全局作用域

```
# 全局作用域优先级高于内置作用域
input = 123  # 把内置函数input覆盖掉了
print(input)
```

全局作用域>内置作用域

内置函数

- 函数名不要与内置函数重名

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

- https://docs.python.org/3/library/functions.html#abs

函数递归

- 递归的递归定义：
  - 递归：参见“递归”
  - 调用自己

```
def accumulate(n):
    if n == 1:
        return 1  # 最小问题，满足这个条件时函数直接返回一个值，退出递归
    else:
        return accumulate(n - 1) + n  # 依次加上从 n 递减到 2 的每个数字


print(f"sum = {accumulate(100)}")
```

```
sum = 5050
```

递归函数

- 优点：有些情况下，可读性比循环更高（幂、阶乘、二分查找……）
- 缺点：效率可能比循环低；可能会栈溢出（超过最大递归深度，耗尽内存）

```python
def recursion():
    print("recursion")
    return recursion()  # 调用自己，且没有退出机制


recursion()  # 无穷递归，必然出错：栈溢出
```

递归接近1000次之后栈溢出

```
  [Previous line repeated 993 more times]
  File "e:\Testfield\summer_python\Lecture_2\L2_recursion.py", line 2, in recursion
    print("recursion")
RecursionError: maximum recursion depth exceeded while calling a Python object
```

匿名函数

- 匿名函数：lambda 表达式
- 写法简便：仅有单条语句组成，省略了def定义，
- 运行效率高：不占用栈内存

```
def judge_PF(score):
    if score >= 60:
        grade = "P"
    else:
        grade = "F"
    return grade


# 改写为匿名函数（lambda 表达式）
f = lambda score: "P" if score >= 60 else "F"
print(judge_PF(61.5))
print(f(58))  # 调用方法基本没有区别
```

第二讲：基本语法结构-总结

课后练习

1.（基本要求）理解并实现以下三段代码，自由地做一些改造：
https://www.runoob.com/python3/python3-calculator.html
https://www.runoob.com/python3/python3-99-table.html
https://www.runoob.com/python3/python-cube-sum.html

2.（基本要求）练习Python list 常用操作：
https://www.runoob.com/python3/python3-list-operator.html

3.（选作）注册Leetcode并做一些“简单”题目：
https://leetcode-cn.com/problems/jewels-and-stones/
https://leetcode-cn.com/problems/guess-numbers/

谢谢大家！

- 反馈问卷