清華大學
Tsinghua University

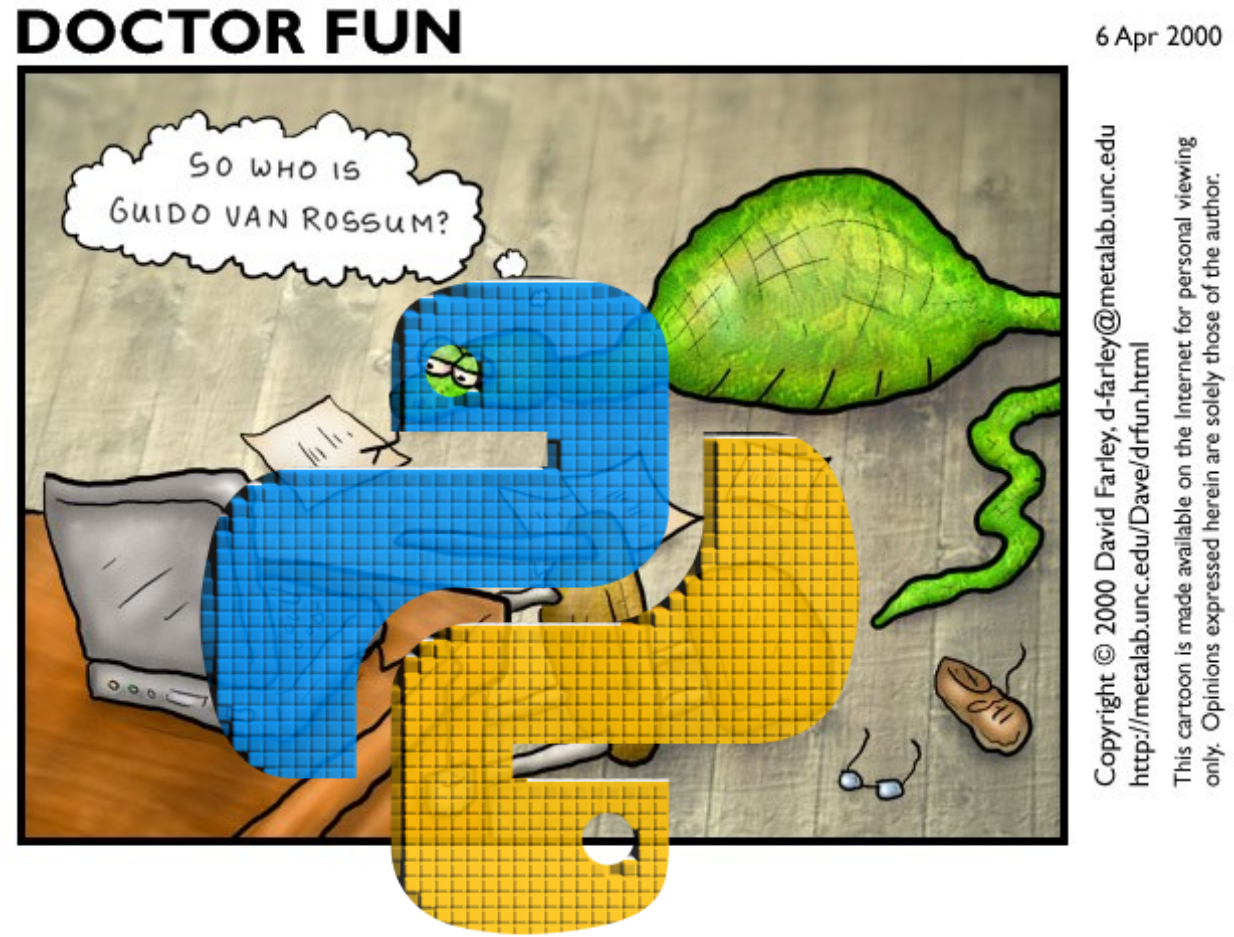
Python 程序设计基础知识介绍

孙鑫礼

2025年10月26日

Python 程序设计语言简介

[荷兰] Guido van Rossum,
the author of Python



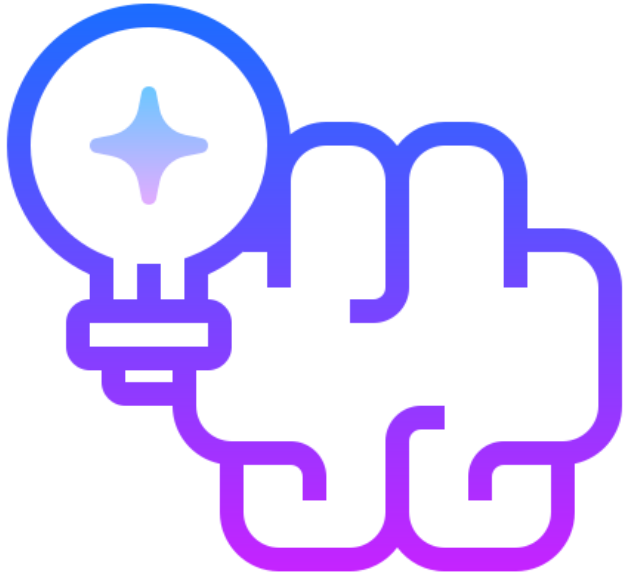
https://gvanrossum.github.io/

Python 程序设计语言简介

Oct 2025	Oct 2024	Change	Programming Language	Ratings	Change
1	1		Python	24.45%	+2.55%
2	4	▲	C	9.29%	+0.91%
3	2	▼	C++	8.84%	-2.77%
4	3	▼	Java	8.35%	-2.15%
5	5		C#	6.94%	+1.32%
6	6		JavaScript	3.41%	-0.13%

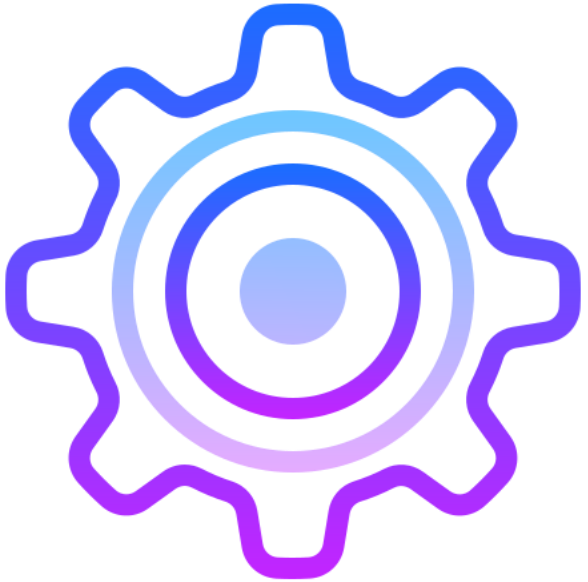
https://www.tiobe.com/tiobe-index/

Python 程序设计语言简介 – 语言特点



语法简洁优雅
适合描述顶层业务逻辑

丰富的第三方库
良好的编程社区

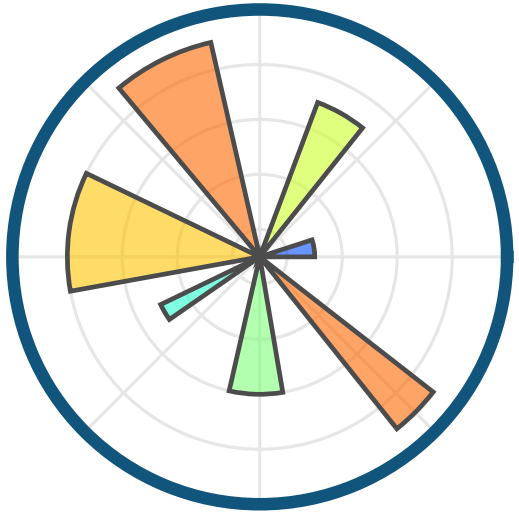


在诸多领域具有广泛应用

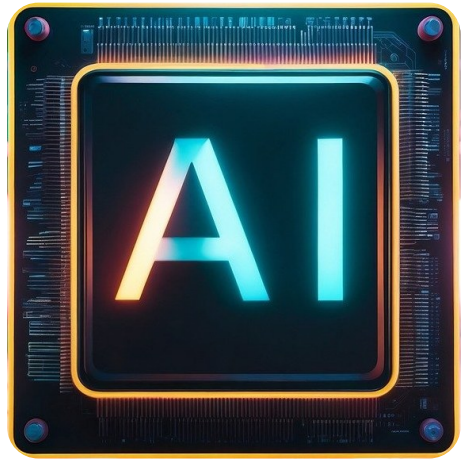
Python 程序设计语言简介 – 应用

办公应用

网络爬虫



科学计算与可视化



人工智能

Python 编程环境配置

- Python是一种解释型语言，可通过Python解释器将Python语言编写的程序代码转化为计算机能够理解的机器指令，并执行这些指令。一般通过IDE进行Python程序开发。
- IDE（Integrated Development Environment，集成开发环境）为编写和调试代码提供了一个统一的界面，可以通过集成多种工具和功能，使开发者能够在一个环境中完成从编码到部署的全过程。
- IDE 一般包含如下关键组件：
  - 编译器或解释器
  - 代码编辑器
  - 调试器
  - 插件系统

Python解释器

Visual Studio Code

PyCharm

- 配置参考：

https://flyingpointer2.github.io/python_learning/NOTES/Python/0_Python_install.md

1. 从第一个 Python 程序开始

```
print("Hello world!")
```

字符串（string）类型的数据；可以单引号、双引号或者三引号包裹

```
print('Hello world!')
print("Hello world!")
print('''Hello world!''')
```

定义一个变量，并将字符串数据绑定到变量上；

```
message = "Hello world!"
print(message)
```

可以通过赋值语句为变量绑定新的数据，这样会覆盖掉旧的数据

```
message = "你好，世界！"
print(message)
```

2. 变量的命名与调用规则

- 通常由英文字母、数字、下划线组成，且不能以数字开头

```
message = "Hello"
message_02 = "world"
价格 = "9磅15便士"
_age = 18
```

- 在Python语言中，“赋值即定义”，不需预先声明变量类型；通过赋值，变量与数据进行绑定或重新绑定

```
id = 2024012345
id = "THU2024"
```

- 注意：不能调用未定义（即未赋值）的变量，否则会引起错误

p2_variables.py

3. 基本数据类型 – 3.1 定义与类型查看

“基本”数据类型

- 整数　int
- 浮点数　float
- 字符串　str
- 布尔型　bool

- 可以用 type 函数查看变量或数据的类型

```
a = "string"
b = 2024
c = 3.14
d = True
e = False

print(type(a), type(b), type(c), type(d), type(e))
```

布尔型的数据仅有 True（真）与 False（假）两种取值

```
<class 'str'> <class 'int'> <class 'float'> <class 'bool'> <class 'bool'>
```

p31_variable_type.py

3. 基本数据类型 – 3.2 整数与浮点数的运算和转化

“基本”数据类型

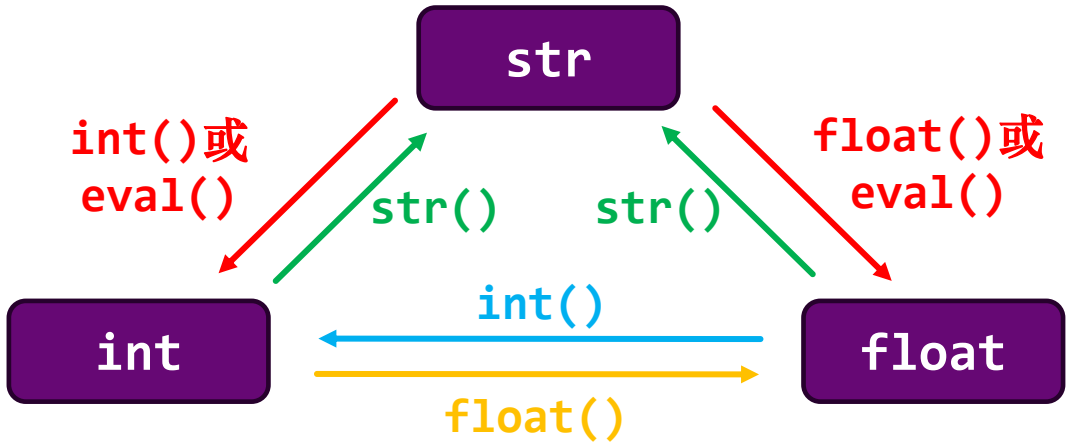
- 整数　　int
- 浮点数　　float
- 字符串　　str
- 布尔型　　bool

- 常用运算符：+，-，*，/，**(乘方)，%(取余)，//(整除)

```
print(100/7)
print(100//7)
print(100%7)
print(3**2024)
```

Python的整数类型支持长整数运算

- 数据类型转化的常用方法：



p32_int_and_float_operation.py

(*)3. 基本数据类型 – 3.2 复数简介

“基本”数据类型

- 整数　int
- 浮点数　float
- 字符串　str
- 布尔型　bool

- 在 Python 中，复数（complex number）是一种内置数据类型。
- 复数的标准形式为 a + bj，其中 a 是实部，b 是虚部，j 表示虚数单位。
- 定义一个复数：

```
z = 3+4j
z = complex(3,4)     # Re(z) = 3, Im(z) = 4
```

- 取复数的实部、虚部、共轭复数、模：

```
print(z,"的实部为",z.real)
print(z,"的虚部为",z.imag)
print(z,"的共轭复数为",z.conjugate())
print(z,"的模（绝对值）为",abs(z))
```

- 常用运算符：+，-，*，/，**(乘方)，与数学中的复数运算规则一致

3. 基本数据类型 – 3.3 字符串 – 运算符

“基本”数据类型

- 整数　int
- 浮点数　float
- 字符串　str
- 布尔型　bool

- 运算符：+用于多个字符串拼接，*用于本字符串重复多次

```
first_name = "Sherlock"
last_name = "Holmes"
full_name = first_name + " " + last_name
print(full_name)
```

```
split_str = 20*"="
print(split_str)
```

- 字符串不能直接和整数、浮点数通过"+"拼接

```
GPA = 3.999
message = first_name + ", your GPA is " + str(GPA)
message = first_name + ", your GPA is " + GPA # Error
```

- 在信息处理中，字符串编码与解码需保持一致（尤其是含中文等非ASCII内容时，编解码不一致会导致乱码错误）
- 常见的编码方式包括 UTF-8、UTF-16、GB2312、GBK、ASCII 等

p33_string_operation.py

3. 基本数据类型 – 3.3 字符串 – 常用方法

“基本”数据类型

- 整数　　int
- 浮点数　　float
- 字符串　　str
- 布尔型　　bool

- 在字符串的处理中，常用的方法包括：

函数	用法
len()	返回字符串的长度
startswith() / endswith()	检查字符串是否以指定的子字符串开始 / 结尾
replace()	将字符串中的某些子字符串替换为另一个子字符串
split()	根据指定的分隔符将字符串分割成多个子字符串，并返回一个列表
join()	将序列中的元素以指定的字符连接成一个新的字符串
upper() / lower() / title()	将字符串中的所有英文字母转换为大写字母 / 小写字母 / 仅单词首字母大写

3. 基本数据类型 – 3.3 字符串 – 常用方法

“基本”数据类型

- 整数　　int
- 浮点数　float
- 字符串　str
- 布尔型　bool

- 举例：

```
msg = "你好，世界！"

print("字符串的长度为 ", len(msg))
print(msg.startswith("你")) # 判断字符串msg开头是否为"你"
print(msg.endswith("世界")) # 判断字符串msg结尾是否为"世界"

msg_replaced = msg.replace("你", "我")
print(msg, msg_replaced)

print(msg.split(", "))    # 默认是空白分隔，也可自定义，如下划线、逗号等
print(" ".join(["How", "are", "you"]))

name = "sherlock HOLmes;"
print(name.upper(), name.lower(), name.title())
```

p33_string_methods.py

3. 基本数据类型 – 3.4 布尔型与条件语句

“基本”数据类型

- 整数　　int
- 浮点数　　float
- 字符串　　str
- 布尔型　　bool

- 除直接赋值外，条件判断语句也会返回布尔型的数据
- 常用的逻辑运算符如下表所示；注意逻辑等于（==）与赋值（=）语句的区别

符号	含义
>	大于
<	小于
==	等于
>=	大于或等于
<=	小于或等于
!=	不等于

3. 基本数据类型 – 3.4 布尔型与条件语句

“基本”数据类型

- 整数　　int
- 浮点数　　float
- 字符串　　str
- 布尔型　　bool

- 多个条件之间的逻辑运算：与（and）、或（or）、非（not）

运算符	条件1	条件2	结果
and	True	True	**True**
	True	False	False
	False	True	False
	False	False	False

“全真才真，一假则假”

运算符	条件1	条件2	结果
or	True	True	True
	True	False	True
	False	True	True
	False	False	**False**

“全假才假，一真则真”

运算符	条件1	结果
not	True	False
	False	True

“取反”

3. 基本数据类型 – 3.4 布尔型与条件语句

“基本”数据类型

- 整数　int
- 浮点数　float
- 字符串　str
- 布尔型　bool

- 示例：

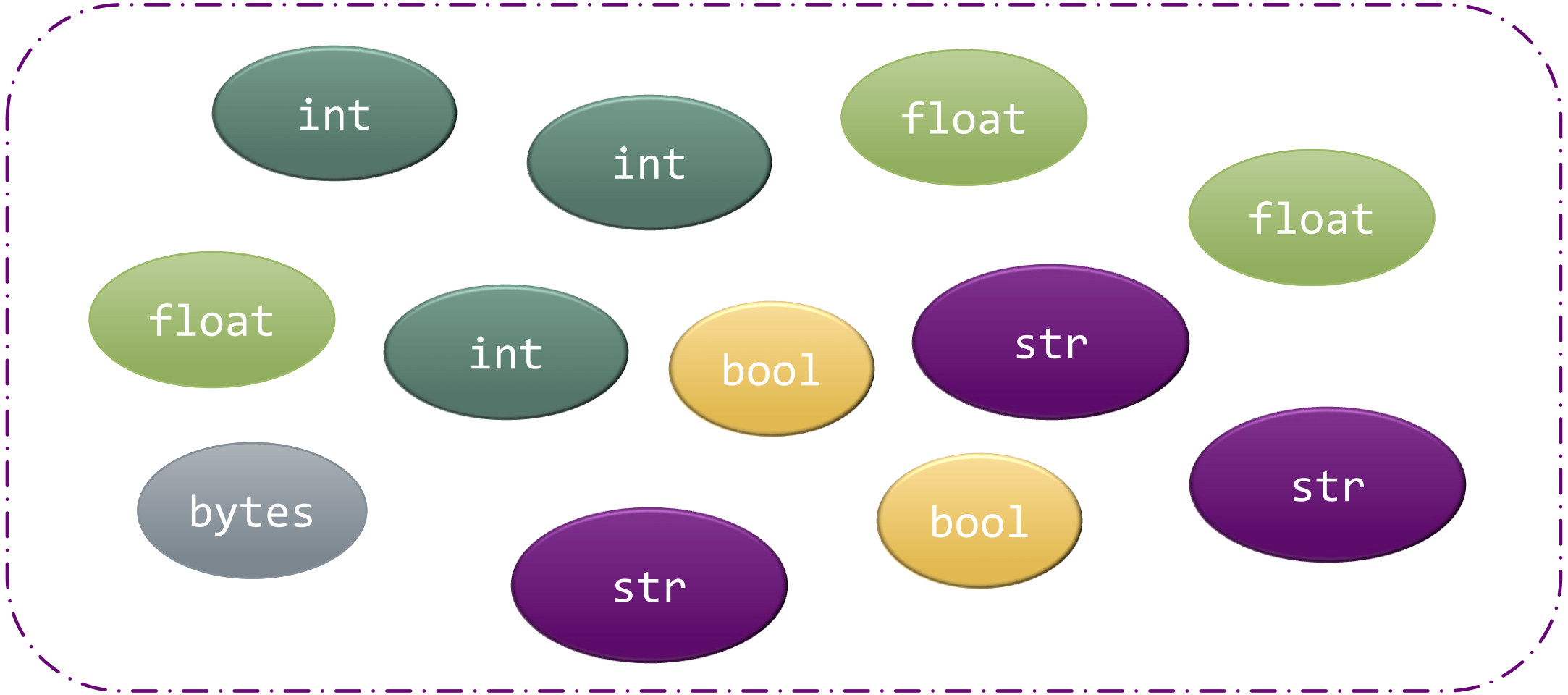
```
year = 2024
print(year, " 是否为偶数？ ", year % 2 == 0)
print(year, " 是否为奇数？ ", year % 2 != 0)
print(year, " 能否被400整除？ ", year % 400 == 0)

GPA = 3.99
conditional_test = (GPA > 3.9 and GPA < 4.0)
print(conditional_test)

year = 1900
condition_1 = year % 400 == 0
condition_2 = year % 4 == 0 and year % 100 != 0
is_gap_year = condition_1 or condition_2
print(year, " 是否为闰年？ ", is_gap_year)
```

```
# 优先顺序：单个条件计算 > 逻辑运算（与或非）> 赋值操作
```

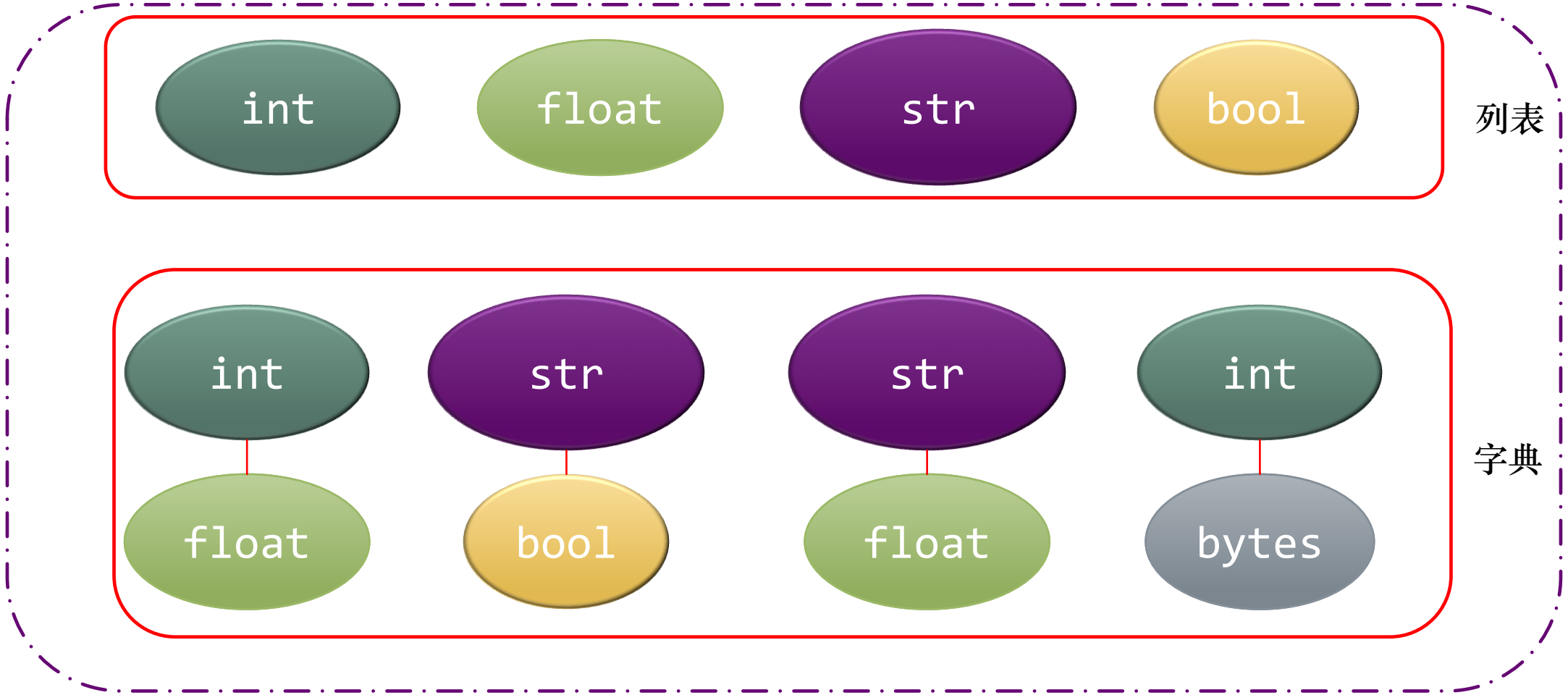
```
#条件1：年份可以被400整除
#条件2：年份能被4整除，但是不能被100整除
```

p34_bool_operations.py

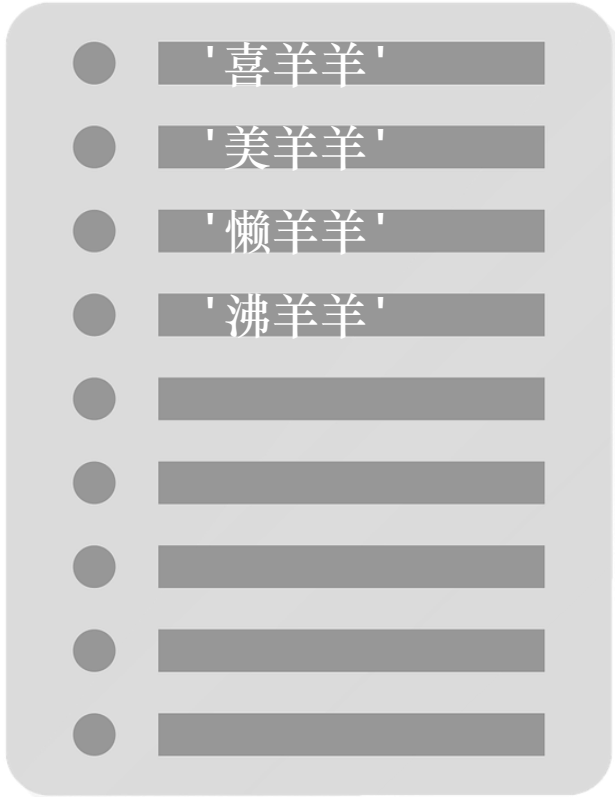
大量数据的组织形式……



大量数据的组织形式……


int
float
str
bool
列表
int
float
str
bool
str
float
int
bytes
字典

大量数据的组织形式……列表（list）



- 列表是按特定顺序排列的一组项目，其中的每一个项目称为列表的一个元素（element）。在Python中，由方括号[]构建一个列表，列表中的各个元素由英文逗号分隔。例如，我们可以创建一个包含羊村中部分羊的名字的列表：

```
names = ['喜羊羊', '美羊羊', '懒羊羊', '沸羊羊']
```

- 列表中的元素不需要相互关联，数据类型可以不同。

```
grades = [95, 'A-', 89, 'A+', 90]
```

- 因为列表通常包含多个元素，所以为列表变量命名时可使用复数形式，如 letters、digits、names。

大量数据的组织形式……字典（dict）

- 字典是一种键值对的集合。每个键（key）都与一个值（value）相关联。在Python中，字典使用大括号 {} 包裹，括号内包含一系列键值对（以冒号分隔），键值对之间用逗号分隔。
- 例如，可以构建如左图所示的“词典”：

```
my_dict = {
    'spaning': 'Nachfor-schung f; ...',
    'spanjor': 'Spanier m; spanjorska ...',
    'spankulera': 'dahin-schlendern, umher...'
    # ......
}
```

大量数据的组织形式……字典（dict）

- 一个键所对应的值可以是Python中的任何数据类型，但键的数据类型需为不可变类型（如字符串、整数等）。例如：

```
students_grades = {
    2024011001: 'A',
    2024011002: 'A-',
    2024011003: 'B',
    2024011004: 'A-',
}
```

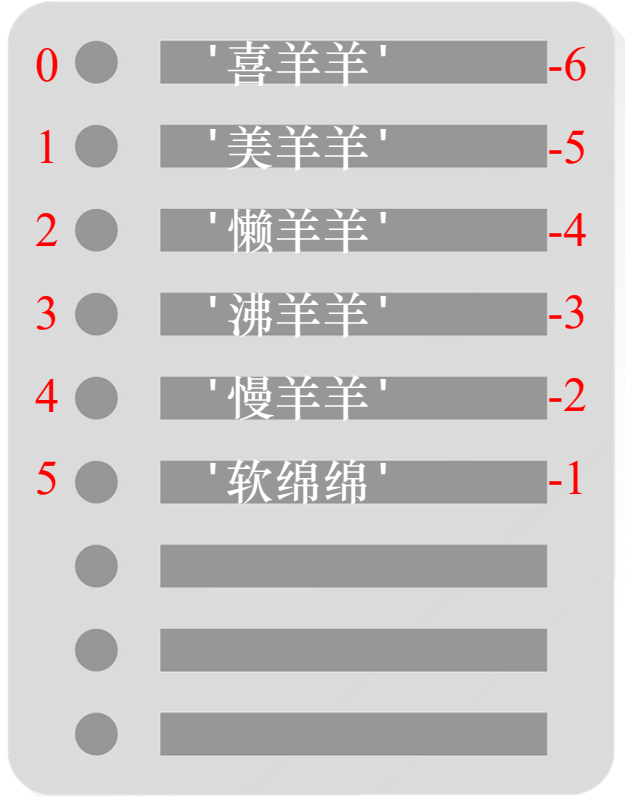
- 可以使用键或get()方法来访问与该键相关联的值。例如：

```
print(students_grades[2024011002])
print(students_grades.get(2024011002))
```

4. 复合数据类型 – 4.1 列表 – 列表的索引与切片

- 列表可以通过整数索引（index）获取对应位置的元素
  - 若索引为非负整数，则首个元素索引为0，从首至尾依次递增
  - 若索引为负整数，则末尾元素索引为-1，从尾至首依次递减

```
names = ['喜羊羊','美羊羊','懒羊羊','沸羊羊','慢羊羊','软绵绵']
print(names[0], names[-1])
```



- 可以通过切片（slice）获取列表中的多个元素组成的子列表
- 切片的一般规则：列表名[初始索引:终了索引:间隔]。间隔默认为1，所形成的子列表中，包含初始索引对应的元素、但不包含终了索引对应的元素（类比数学上的“前闭后开区间”）

```
names = ['喜羊羊','美羊羊','懒羊羊','沸羊羊','慢羊羊','软绵绵']
```

1 2 3 4

```
names[1:4]
```

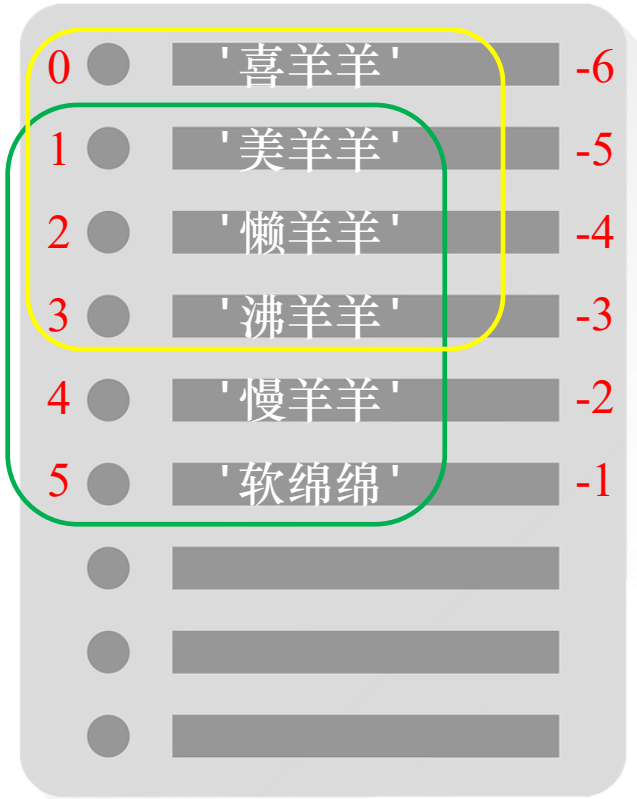
注意：不包含终了索引

4. 复合数据类型 – 4.1 列表 – 列表的索引与切片

- 切片的一般规则：列表名[初始索引:终了索引:间隔]。间隔默认为1，所形成的子列表中，包含初始索引对应的元素、但不包含终了索引对应的元素。

```
names = ['喜羊羊','美羊羊','懒羊羊','沸羊羊','慢羊羊','软绵绵']
```

```
names[1:4:2]        1        2        3        4
```



- 初始索引缺省为0，终了索引缺省为末尾元素索引+1。例如：

```
print(names[1:])   # 包含索引1、2、3、4、5所对应的元素
print(names[:4])   # 包含索引0、1、2、3所对应的元素
print(names[:])    # 包含索引0、1、2、3、4、5所对应的元素（即整个列表）
```

- 间隔值可以为负整数，表示切片是按（源列表）从后向前的顺序排列。例如：

```
print(names[::-1]) # 逆序排列
```

```
['软绵绵', '慢羊羊', '沸羊羊', '懒羊羊', '美羊羊', '喜羊羊']
```

4. 复合数据类型 – 4.1 列表 – 列表的常用方法

- 列表常用方法

函数	用法	
len()	获取列表的长度（即列表中元素的数量）	
append()	在列表末尾追加一个元素	增
extend()	将另一个列表中的所有元素追加到当前列表末尾	
pop()	移除并返回列表中指定位置的元素；如果不指定索引，默认移除并返回列表的最后一个元素	删
remove()	用于删除列表中的第一个指定值；如果不存在该值，会抛出错误	
insert()	在指定索引处插入元素	插入数据
index() / in	查找元素的索引 / 判断元素是否在列表中	查找数据
sort() / sorted()	对列表进行原地排序，改变原列表 / 对列表进行排序，不改变原列表，返回排序后的新列表	数据排序

p41_list_methods.py

(*) 4. 复合数据类型 – 4.1 列表 – 作用于列表的“+”与“*”运算符

- 列表与列表之间可用“+”运算符进行连接，其作用近似于extend()方法。例如：

```
goats = ['喜羊羊', '美羊羊', '懒羊羊', '沸羊羊']
wolves = ['红太狼', '灰太狼']

goats_and_wolves = goats + wolves
print(goats_and_wolves)
```

```
['喜羊羊', '美羊羊', '懒羊羊', '沸羊羊', '红太狼', '灰太狼']
```

- 列表与正整数之间可以用“*”运算符，表示列表重复多次。例如：

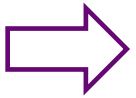
```
three_group_of_wolves = wolves * 3
print(three_group_of_wolves)
```

```
['红太狼', '灰太狼', '红太狼', '灰太狼', '红太狼', '灰太狼']
```

4. 复合数据类型 – 4.2 字典 – 字典的键、值与键值对

- 可以通过 keys(), values(), items()方法来查看一个字典有哪些键、值以及键值对
- 字典和列表可以嵌套，如字典的值可以是列表或字典，列表的元素可以是字典或列表，等等。例如：

学号	姓名	成绩		
		微积分	大雾	小雾
2024999001	Em ily	84	77	71
2024999002	Jacob	92	77	78
2024999003	Sarah	88	85	76
2024999004	M ichael	80	82	85
2024999005	Laura	78	71	71
2024999006	Kevin	88	73	91
2024999007	O livia	71	89	87
2024999008	Ethan	61	77	72
2024999009	Am anda	88	81	91
2024999010	Nathan	83	68	65



```
students_info = {
    2024999001:{
        '姓名':'Emily',
        '成绩':{'微积分':84, '大雾':77, '小雾':71}
    },
    2024999002:{
        '姓名':'Jacob',
        '成绩':{'微积分':92, '大雾':77, '小雾':78}
    },
    # ......
}
```

p42_dict-1.py

4. 复合数据类型 – 4.2 字典 – 字典的键、值与键值对

- 可通过键获取或修改该键对应的值。例如：

```
print('学号2024999002的同学的姓名是', students_info[2024999002]['姓名'])
print('这位同学的微积分成绩是', students_info[2024999002]['成绩']['微积分'])
```

```
students_info[2024999002]['成绩']['微积分'] = 95
```

- 可通过键增加新的键值对。例如，新增学号2024666666的同学信息：

```
students_info[2024666666] = {
    '姓名':'You know who',
    '成绩':{'微积分':66,'大雾':66,'小雾':66}
}
```

- 可通过 del 删除已有的键值对。例如，删除学号2024999002的同学信息：

```
del students_info[2024999002]
```

```
students_info = {
    # ......
    2024999002:{
        '姓名':'Jacob',
        '成绩':{'微积分':92,
                '大雾':77,
                '小雾':78
               }
    },
    # ......
}
```

p42_dict-2.py

4. 复合数据类型 – 4.3 其他复合数据类型简介 – 元组（tuple）

- 元组（tuple）是一种不可变的序列类型，由一对小括号()包裹，其中的元素以逗号分隔。
- 元组的索引、切片规则与列表一致；但元组的内容不能修改，即不能添加、删除或修改其中的元素，除非用新的元组数据完全覆盖旧的元组。
- 可以通过索引或切片访问元组中的元素，其规则与列表一致。例如：

```
point_tuple = (3, 4, 5)
print(point_tuple[0])
print(point_tuple[0:2])
```

- 列表可以通过索引或切片修改其中的元素，但元组不能。例如：

```
point_list = [3, 4, 5]
point_list[2] = 99
```

```
point_tuple[2] = 99
        # TypeError: 'tuple' object does not support item assignment
```

(*) 4. 复合数据类型 – 4.3 其他复合数据类型简介 – 集合（set）

- 集合是一个无序的、不包含重复元素的集合数据类型，由一对花括号{}包裹，其中的元素以逗号分隔。
- 可以从列表或元组创建集合，创建后的集合中，将不包含重复数据。例如：

```
my_set = set([1,1,2,3,5,13,34,3])
print(my_set)
```

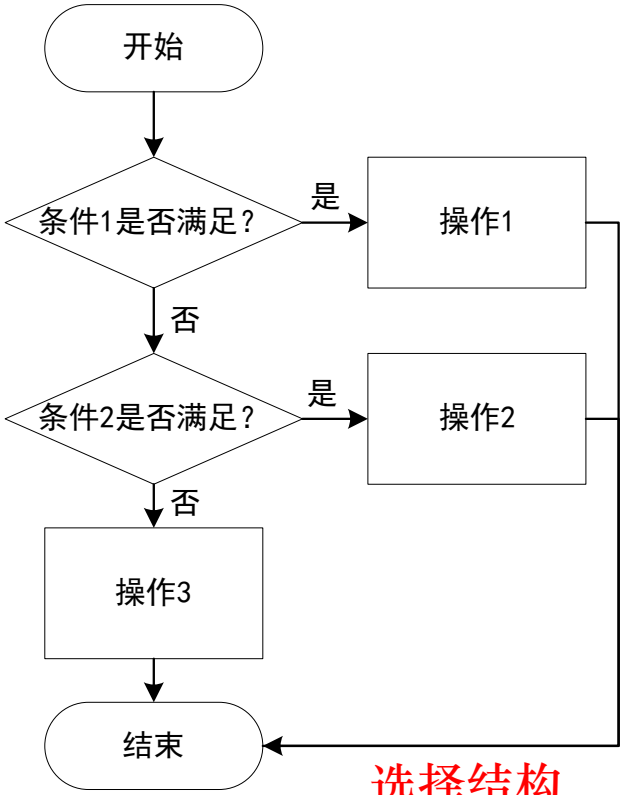
```
{1, 2, 3, 34, 5, 13}
```

- 集合的常用方法：

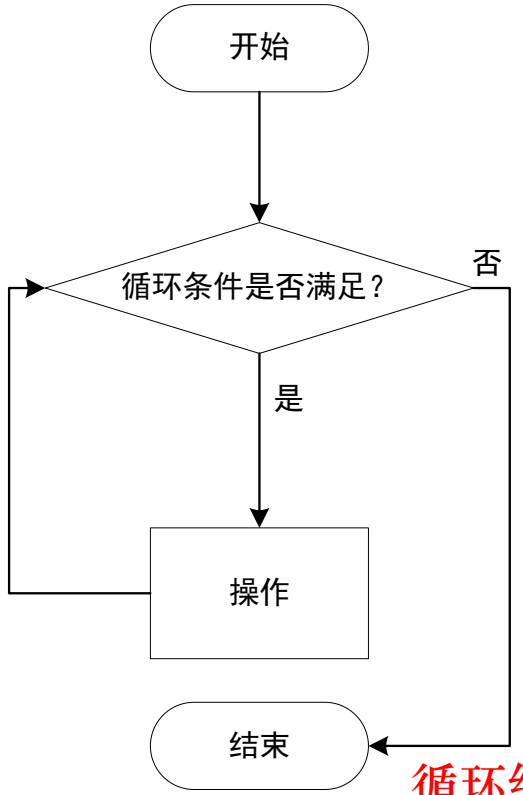
函数	用法
len()	获取集合的长度（即集合中元素的数量）
add()	向集合中添加元素
discard()	删除集合中的元素
set1.union(set2), set1 | set2	返回两个集合的并集
set1.intersection(set2), set1 & set2	返回两个集合的交集
set1.difference(set2), set1 - set2	返回两个集合的差集

5. 流程控制

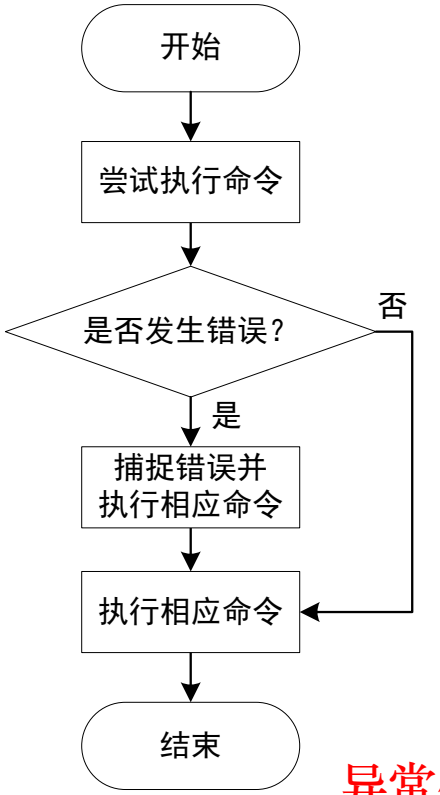
- 前文中给出的Python程序都是顺序执行的，即Python解释器从上到下逐行执行代码，遇到错误则报错并终止程序执行。除此之外，Python中还有其他特定的语法结构来控制程序的执行顺序和逻辑，如选择结构、循环结构、异常处理等。



选择结构

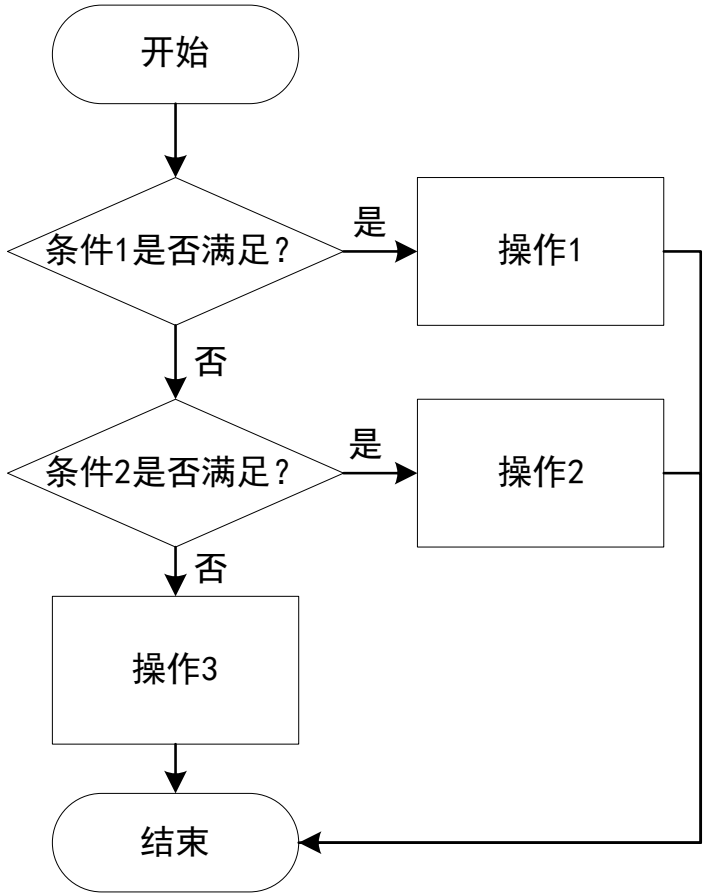


循环结构



异常处理

5. 流程控制 – 5.1 选择结构



- 选择结构通过if，else，elif等关键字进行控制；当仅有一个条件判断时，可以省略elif关键字。例如，判断一个年份是否为闰年：

```
year = 2024

if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
    print(year, "年是闰年")
else:
    print(year, "年不是闰年")
```

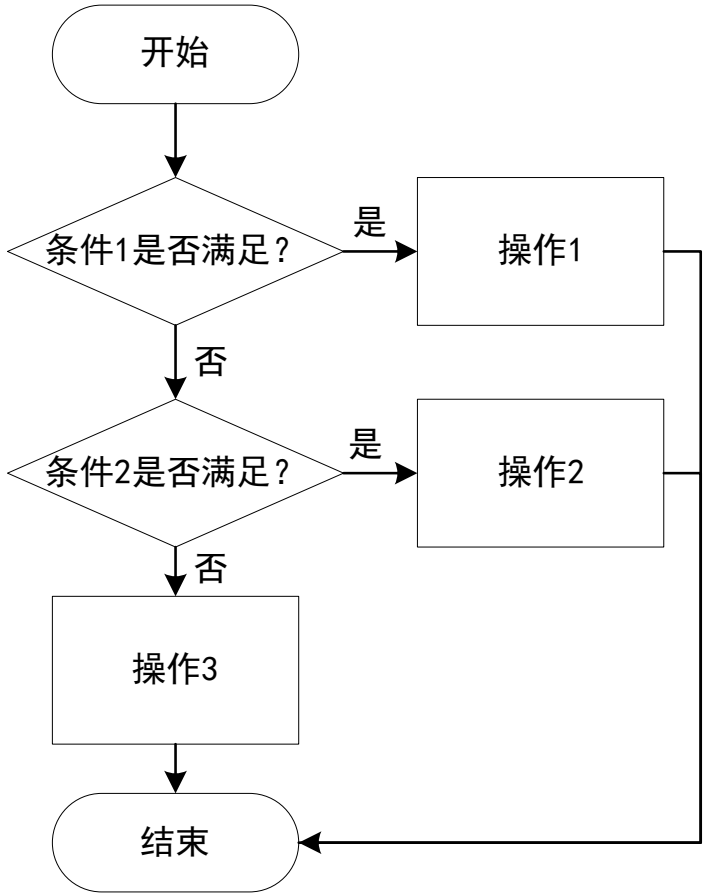
条件判断（结果是True或False）

若上述条件成立（即结果为True），则执行此处的语句块

若上述条件不成立（即结果为False），则执行此处的语句块

p51-if-structure-1.py

5. 流程控制 – 5.1 选择结构



- 当存在多个条件判断时，可使用elif构成选择结构。例如，左图的流程结构可等效为如下的Python语句：

```
if condition_1:
    print("条件1满足！")
    print("执行操作1")
elif condition_2:
    print("条件1不满足，但条件2满足！")
    print("执行操作2")
else:
    print("条件1、条件2都不满足！")
    print("执行操作3")
```

每个条件判断以及else后须接冒号

每个条件判断对应的语句块需整体缩进

p51-if-structure-2.py

5. 流程控制 – 5.1 选择结构 – 缩进设置

- 在Python代码中，不同层次结构（如选择结构、循环结构中的语句块）以缩进进行区分；因此，代码缩进是强制要求。这一点与C++等语言不同。请看如下的对比：

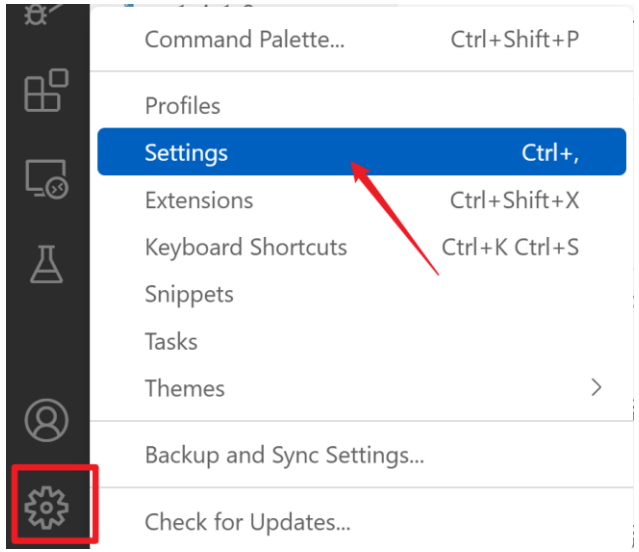
```
year = 2024
if year % 400 == 0 or (year % 4 == 0 and year % 100 != 0):
    print(year, "年是闰年")
else:
print(year, "年不是闰年")
```

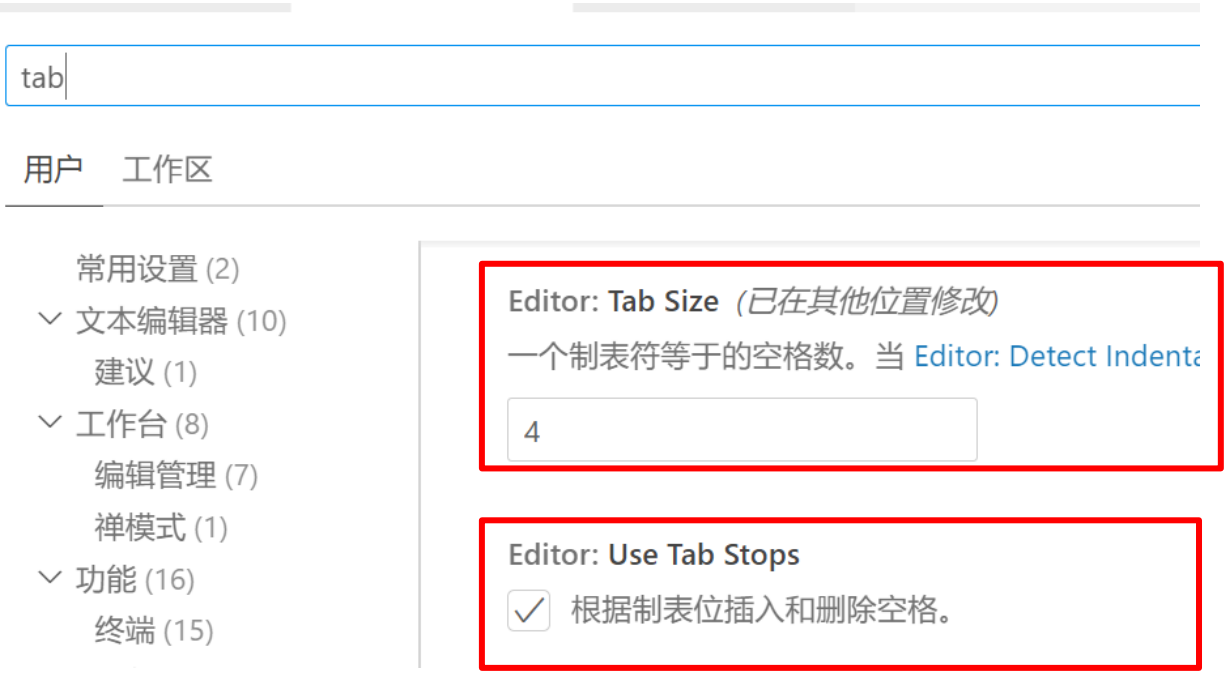
此处的语句块未正确缩进，会导致错误

```
    int year = 2024;
    if (year % 400 == 0 or (year % 4 == 0 and year % 100 != 0)) {
        std::cout << year << " is a gap year." << std::endl;
    }
    else {
std::cout << year << " is NOT a gap year." << std::endl;
    } // C++语言中 可通过花括号{}确定层次结构 对缩进无强制要求
```

5. 流程控制 – 5.1 选择结构 – 缩进设置

- 缩进格式：Tab 键或固定数目的空格
- PEP 8（Python官网给出的编程规范）建议使用四个空格作为缩进规范；在VSCode中，可通过如下方式设置缩进。





https://peps.python.org/pep-0008/#indentation

5. 流程控制 – 5.2 循环结构 – for循环与可迭代对象

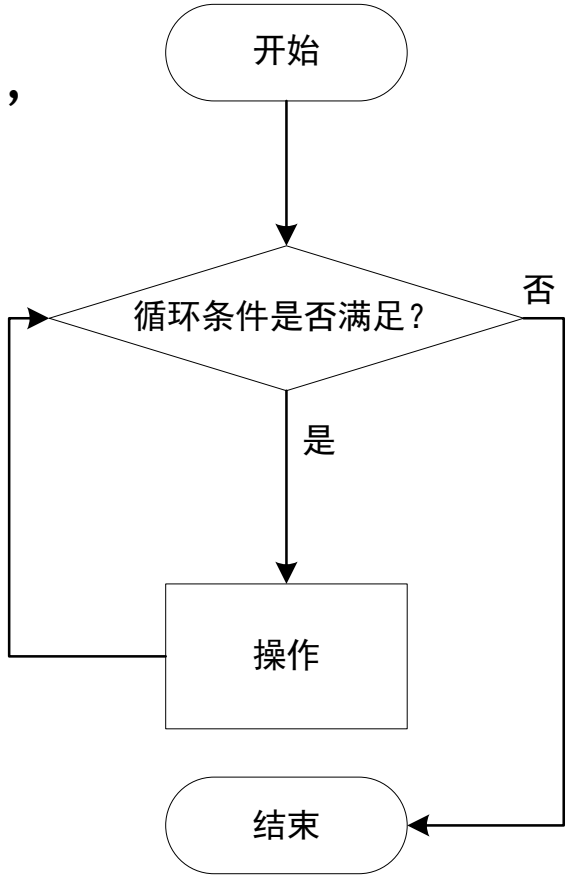
- 在Python语言中，for循环和while循环是两种常见的循环结构。
- for循环主要用于遍历可迭代对象（如列表、元组、字典、字符串等）的元素，常用于预先知道循环次数或者要遍历的数据集的情况。
- for循环的基本语法：

```
for 元素 in 可迭代对象:
    循环体
```

循环条件后须接冒号

循环体的语句块需要整体缩进

若循环条件成立（即元素在可迭代对象中），则执行循环体中的语句；否则退出循环，执行后续的语句



- 例如，可通过下述循环欢迎羊村中的羊来到三体世界（遍历列表）

```
names = ['喜羊羊', '美羊羊', '懒羊羊', '沸羊羊']
for name in names:
    print("你好，" + name + "，欢迎来到三体世界！")
```

p52_for_loop-1.py

5. 流程控制 – 5.2 循环结构 – for循环与range对象

- 在for循环中，可以用range对象来控制循环的次数。
- 可以用range()方法生成range对象，其基本语法为：

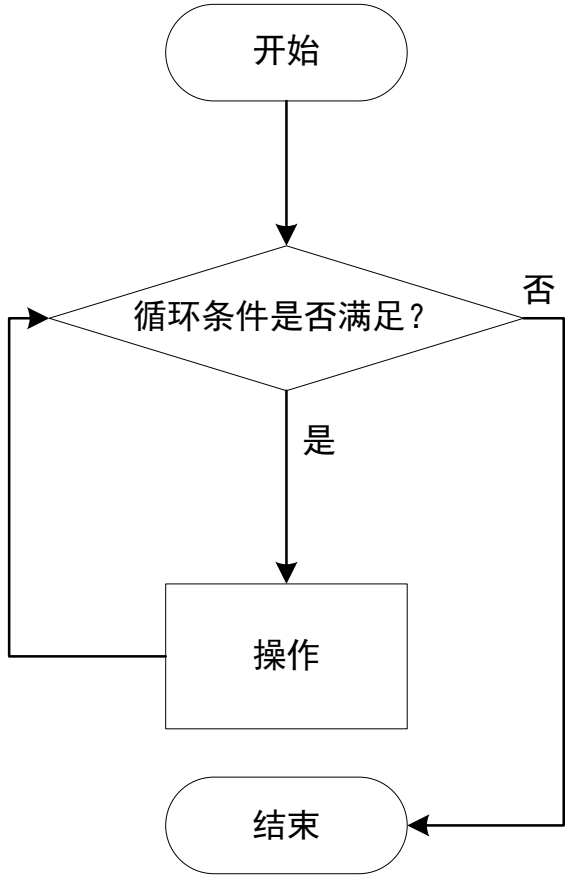
```
range(初始值，终了值，间隔)
```

- 其中，初始值的默认值为0，间隔的默认值为1；终了值不可省略。
- 与列表的切片规则类似，range()方法生成的序列也是“前闭后开区间”，即包含初始值，但不包含终了值。例如：

```
range(5)          # 0,1,2,3,4
range(1,5)        # 1,2,3,4
range(1,9,2)      # 1,3,5,7（间隔为2，且不包含终了值9）
```

- 应用示例：求1~100（闭区间）的整数和

```
result = 0
for number in range(1,100+1,1):
    result += number      等价于：result = result + number
```



p52_for_loop-2.py

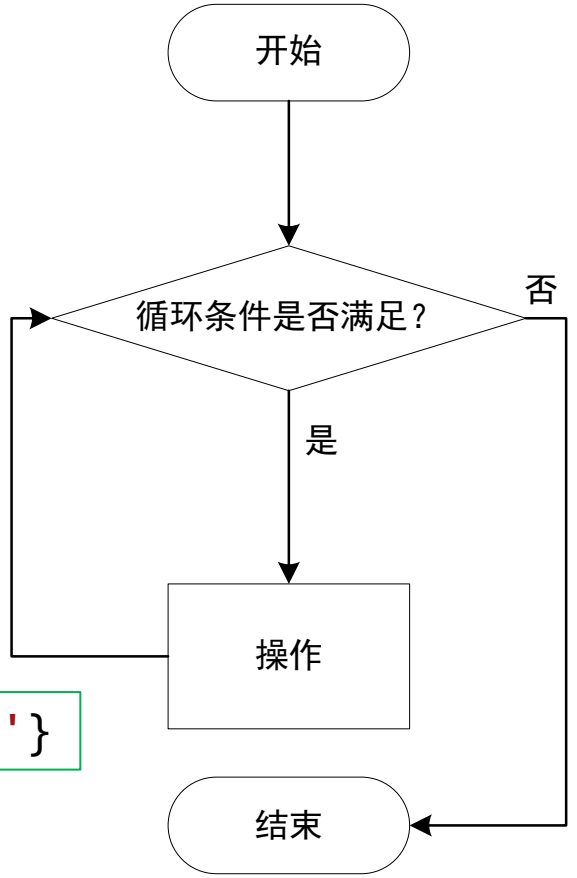
5. 流程控制 – 5.2 循环结构 – for循环遍历字典

- 可以通过 keys(), values(), items()方法来遍历一个字典的键、值、键值对
- 例如：

```
for student_id in students_grades.keys():          # 遍历字典的键
    print("ID = " + str(student_id))
for student_grade in students_grades.values():     # 遍历字典的值
    print("grade = " + str(student_grade))
for stu_id, stu_grade in students_grades.items(): # 遍历字典的键值对
    print("ID = " + str(stu_id) + ", grade = " + str(stu_grade))
```

两个变量分别取得键、值

```
students_grades = {2024011001: 'A', 2024011002: 'A-', 2024011003: 'B'}
```



p52_for_loop-3.py

(*) 5. 流程控制 – 5.2 循环结构 – for循环生成列表（list comprehension）

- 使用for循环结合列表推导式（list comprehension）可以高效、优雅地生成列表。其基本语法为：

```
[表达式 for 元素 in 可迭代对象 if 条件]
```

- 其中，`if 条件`部分可以省略。
- 示例1：1~10（闭区间）范围内的所有整数的平方

```
square_numbers = [i**2 for i in range(1,11)]
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- 示例2：1~10（闭区间）范围内的所有偶数的平方

```
square_numbers = [i**2 for i in range(1,11) if i%2 == 0]
```

```
[4, 16, 36, 64, 100]
```

- 列表推导式的执行时间一般会比等效的显式语句循环更短，在列表生成上更加高效

5. 流程控制 – 5.2 循环结构 – while循环

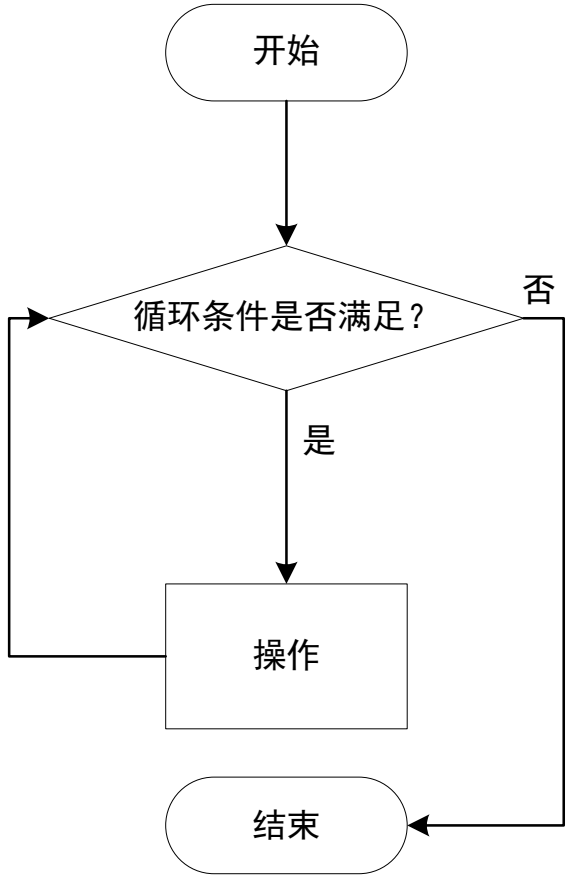
- 在Python中，while循环是基于条件的循环（只要条件为真，循环就会继续执行），常用于不确定需要执行循环的次数、但需要在某个条件满足时继续进行循环的情况。
- while循环的基本语法：

```
while 条件表达式:
    循环体
```

循环条件后须接冒号

循环体的语句块需要整体缩进

若循环条件成立，则执行循环体中的语句；否则退出循环，执行后续的语句



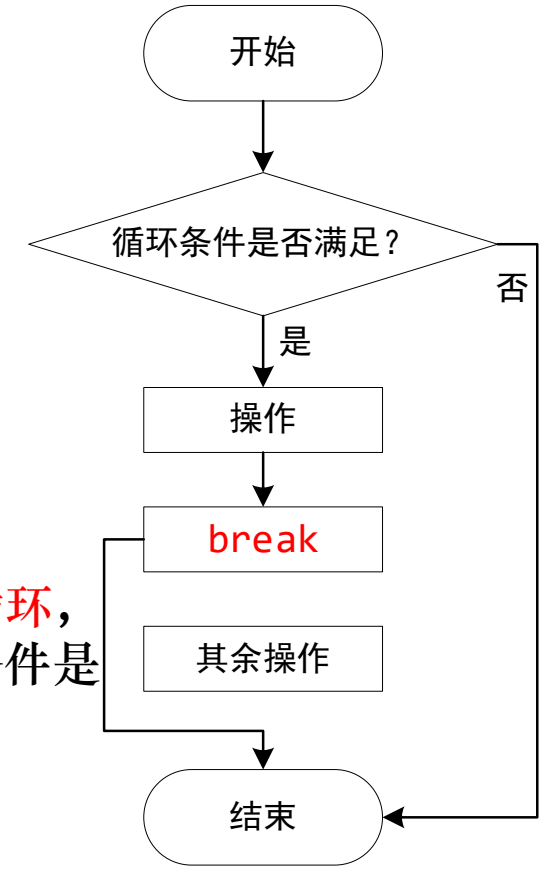
- 示例：牛顿法迭代计算一个正实数a的算术平方根
- 当第(n+1)次迭代值x_{n+1}与上一次迭代值x_n相差足够小时，迭代收敛

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

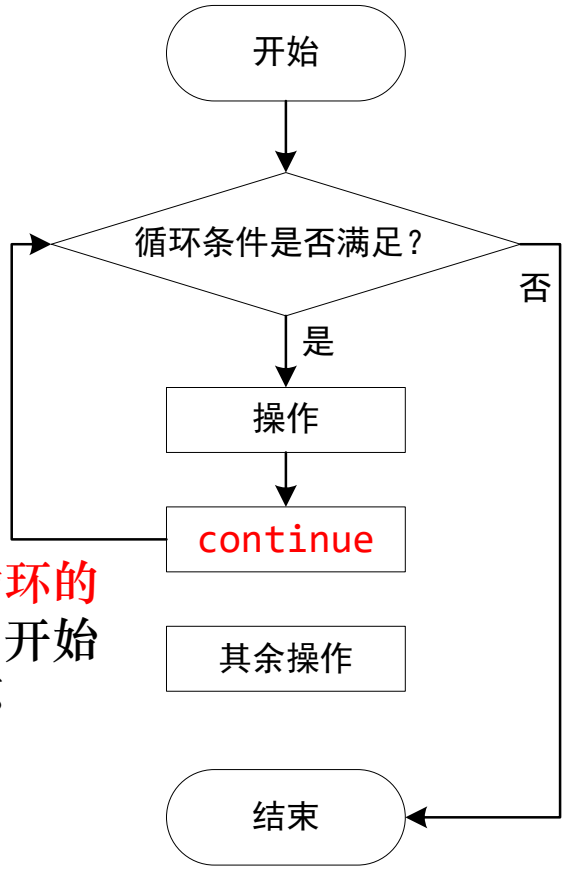
p52_while_loop.py

5. 流程控制 – 5.2 循环结构 – break与continue

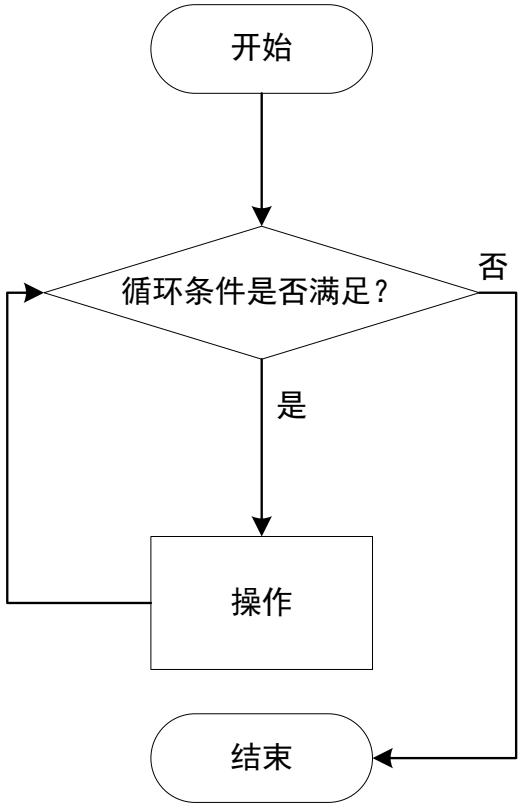
- 在Python中，可用break和continue控制循环的执行流程



终止当前循环，无论循环条件是否仍满足

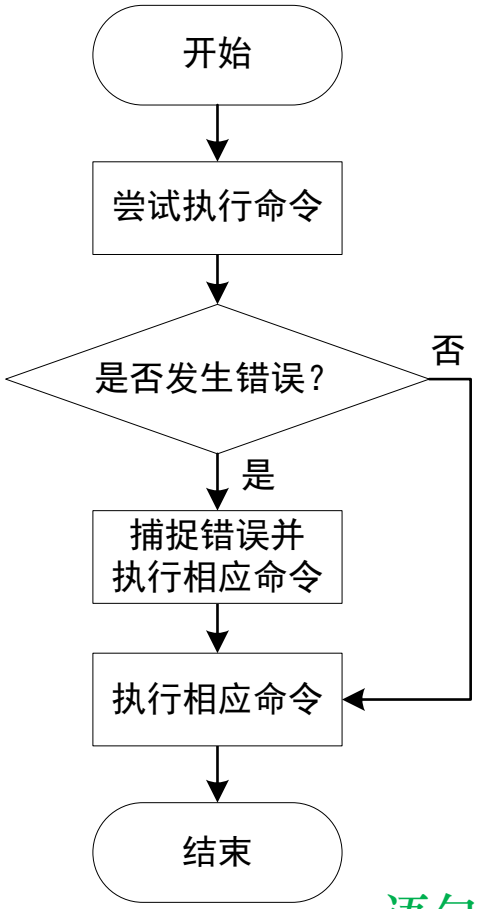


跳过当前循环的剩余部分，开始下一次循环



p52_break.py
p52_continue.py

5. 流程控制 – 5.3 异常处理



- 在Python中，当程序在执行过程中遇到错误时，如无异常处理，程序会终止运行并显示Traceback。例如：接受用户输入并计算其倒数，若用户输入为0，程序会直接报错并终止运行。

```
number = float(input("请输入一个正实数: "))
print("1/{:.3f} = {:.3f}".format(number, 1/number))
```

- 缺点：直接终止程序运行可能会干扰任务实现，安全性和用户友好性不足
- 在Python中，可以用try...except...捕获错误信息。例如：

```
try:                                    此处需加冒号
    print("1/{:.3f} is {:.3f}".format(number, 1/number))
except ZeroDivisionError:
    print("Error!")
    print("You should not input ZERO!")
```

程序首先尝试执行try语句块的命令

语句块需整体缩进

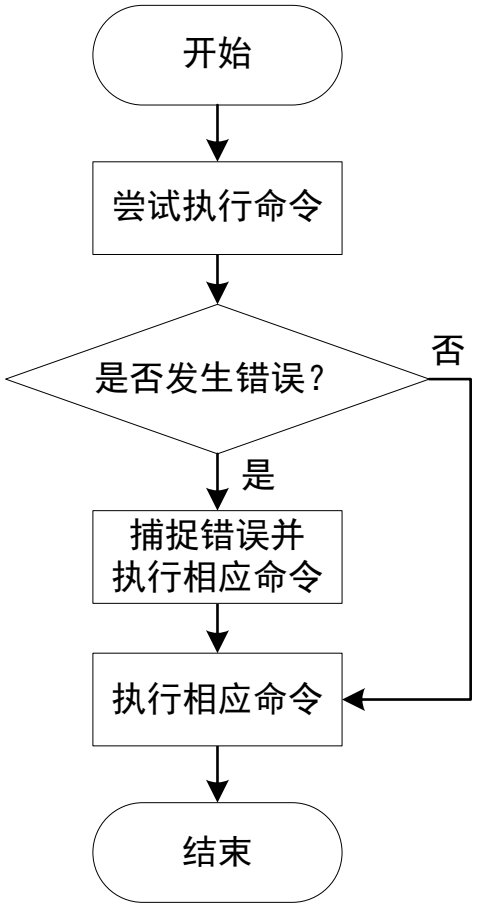
如果遇到错误，则跳转到except语句块

p53_error_handle.py

5. 流程控制 – 5.3 异常处理 – 常见异常类型

异常类型	常见情景
SyntaxError	语法错误，例如使用不存在的操作符、变量名称不符合规则等
NameError	访问未定义的变量
TypeError	操作或函数应用于不适当类型的对象，例如“+”将字符串和浮点数直接连接
IndexError	索引超过序列（如列表、元组等）的有效范围
KeyError	访问字典中不存在的键
ValueError	操作或函数应用于具有类型正确、但值不适当的数据，例如将字符串“1oo”直接用int()函数转化为整数、用remove()函数移除不存在的列表元素
ZeroDivisionError	除法运算时除数为零
FileNotFoundError	尝试访问的文件或目录不存在
AttributeError	访问对象的不存在的属性或方法，例如对整数数据对象使用reverse()函数
ImportError	用import语句导入不存在或尚未安装的模块

5. 流程控制 – 5.3 异常处理

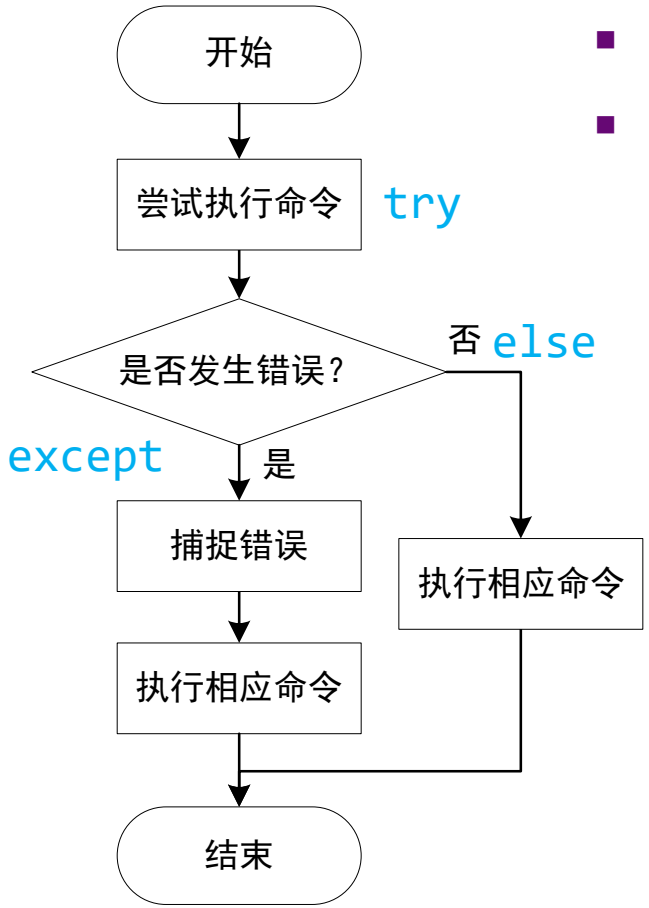


- 如果程序运行中可能产生不同类型的错误，可以分别针对这些错误类型分别在except语句块进行处理。例如：

```
while True:
    user_input = input("请输入一个正实数（输入'q'退出程序）：")
    if user_input == 'q':
        break
    try:
        number = float(user_input)
        print("1/{:.3f} is {:.3f}".format(number, 1/number))
    except ZeroDivisionError:
        print("错误！应该输入非零数字！")
    except ValueError:
        print("错误！应该输入数字！")
```

p53_try_except.py

(*) 5. 流程控制 – 5.3 异常处理



- 异常处理也可以用 try...except...else...进行。
- 程序首先尝试执行try语句块的命令，如果遇到错误，则跳转到except语句块；若正常执行，则跳转到else语句块。例如：通过输入文件读取数据，若文件未找到，则使用默认值。

```
try:
    with open("initial_speed.dat","r") as f:
        input_data_str = f.read()
except FileNotFoundError:
    print("未找到输入文件，初始速度使用默认值")
    v0 = 1.0
else:
    v0 = float(input_data_str)
```

else语句块中可继续嵌套try-except语句

6. 输入输出 – 标准输入输出

- 在Python中，标准输入和标准输出分别通过input()和print()函数实现
  - 标准输入（Standard Input）来自系统默认的输入设备，一般而言是键盘；
  - 标准输出（Standard Output）输出到系统默认的输出设备，一般而言是显示器终端。
  - 程序运行时可以从标准输入获取用户输入；程序执行结果可以通过标准输出显示给用户。
- 标准输入函数 input() 的基本语法：

```
result = input(prompt_message)
```

- 其中，prompt_message是提示信息，为字符串类型；返回值result是用户输入的内容，也是字符串类型。
- 标准输入和标准输出可以被重定向。例如，标准输入可以从文件读取，标准输出也可以重定向到文件。

p6_std_IO.py

6. 输入输出 – 文本文件读写

- 在Python中，可通过 `with open ... as ...` 语句实现文本文件的读写，其基本语法为：

```
with open(file_path, mode='r', encoding=None) as file:
    文件读写操作
```

with语句后须接冒号

文件读写操作的语句块需整体缩进

文件路径（字符串变量）

可选参数，指定文件打开的模式、编码方式等

- 文件读写操作中的常用方法：

方法	用法
file.read()	读取整个文件的内容
file.readlines()	读取文件的所有行，并返回一个包含每行内容的列表
file.write(content)	向文件写入字符串类型的数据
file.writelines(content)	向文件写入字符串列表

p6_file_IO.py

6. 输入输出 – 文本文件读写

- 文件打开模式选项：

模式	描述
'r'	以读取模式打开文件（默认）。文件必须存在。
'w'	以写入模式打开文件。文件不存在则创建，存在则覆盖。
'a'	以追加模式打开文件。文件不存在则创建，存在则从末尾写入。
'r+'	以读写模式打开文件。文件必须存在。
'b'	以二进制模式打开文件。可以与其他模式组合使用（如 'rb'）。

- 文件路径说明：
  - 可以是相对路径或绝对路径
  - 若目录与文件之间以“\”连接（Windows系统），则文件路径变量中需替换为“\\”，或使用raw-string，或替换为“/”，以免被误认为转义字符。

```
file_path_1 = r'C:\Users\point\Downloads\address.csv'
file_path_2 = 'C:\Users\point\Downloads\address.csv'  # SyntaxError
file_path_3 = 'C:\\Users\\point\\Downloads\\address.csv'
```

6. 输入输出 – 文本文件读写

- 在Python中，转义字符是指通过在字符前面添加反斜杠 \ 来赋予这个字符特殊意义的字符。转义字符常用于插入那些在字符串中无法直接包含的字符，例如，换行符、制表符、引号等。
- 常用转义字符：

转义字符	描述	示例
\n	换行符	"Hello\nWorld" ：两行显示 Hello 和 World
\t	水平制表符（Tab）	"Hello\tWorld" ：在 Hello 和 World 之间插入一个 Tab 键
\\	反斜杠	"Escape using \\\\" ： 显示 Escape using \\
\'	单引号	'It\'s easy' ：显示 It's easy
\"	双引号	"She said, \"Hello\"" ：显示 She said, "Hello"

6. 输入输出 – 输出格式控制

- 在Python中，整数、浮点数的标准输出（或写入文件）时，可以进行格式控制。
- 常用格式控制方法包括： %方法、 format()方法、 f-strings方法(Python >= 3.6)
- 例如：

整数　小数位数　浮点数

```
n = 7
print("10/{:d} = {:.2f}".format(n,10/n))     # 保留2位小数
print("10/{:03d} = {:6.2f}".format(n,10/n))    # 整数宽度3、填充0；保留2位小数、宽度为6;
print("10/{:d} = {:.2e}".format(n,10/n))     # 保留2位小数的科学计数法
print("10/{:d} = {:.2}".format(n,10/n))      # 保留2位有效数字
```

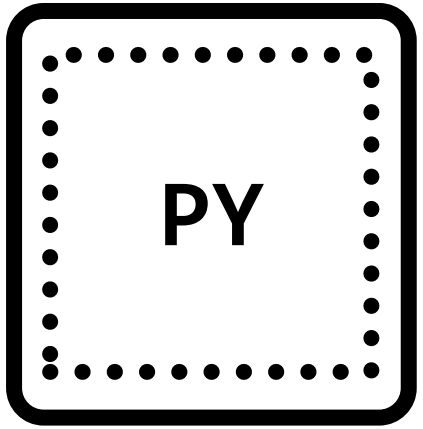
科学计数法表示浮点数

https://docs.python.org/zh-cn/3/library/string.html#formatspec

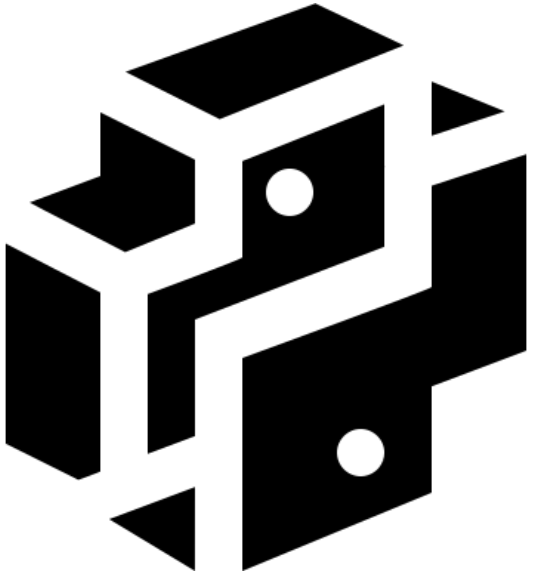
p6_output_format.py

Python 中的模块化

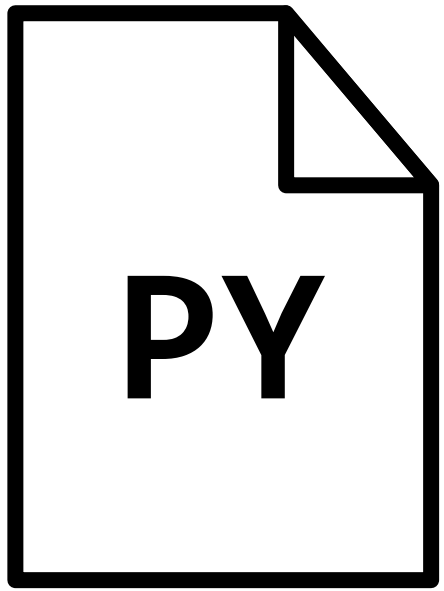
- 在Python中，可运用“模块化”的思想进行代码组织，以提高代码的复用性、可扩展性和可维护性。
- 在Python中，模块化通常涉及函数（function）、类（class）和模块（module）。



函数（function）



类（class）



模块（module）

7. 函数 – 函数定义

- 在Python中，可以将实现特定任务或具有特定功能的代码块封装为函数（function）。
- 函数通过def关键字定义。在定义函数时，需指定函数名、参数列表（可以为空，但括号不能省略）、返回值（可以无返回值）。例如，通过如下代码定义一个计算三角形面积的函数：

```
def triangle_area(a,b,c):
    '''计算三角形的面积'''
    p = (a+b+c) / 2
    S = (p*(p-a)*(p-b)*(p-c))**(1/2)
    return S
```

函数的doc_string，对函数功能和使用方法进行说明

函数体内的语句块需要整体缩进

- 在Python中，直接通过函数名调用函数。例如，对于上述定义的函数，通过下面的语句调用：

```
line_lengths = [3,4,5]
area = triangle_area(line_lengths[0],line_lengths[1],line_lengths[2])
print(f"三角形的三边长分别为面积为{area}")
```

p7_func_define.py

7. 函数 – 函数的实参和形参

- 注意到函数 triangle_area 定义时的参数列表与调用函数的参数列表并不完全相同，前者称为形参（形式参数，parameter），后者称为实参（实际参数，argument）。
- 对比：形参在函数定义中起到标识的作用，表明**函数可以接受什么样的数据**，并提供一个名称以便在函数体中使用。实参是指在调用函数时，**传递给形参的实际值**，是函数执行时的具体数据。

```
def triangle_area(a,b,c):
    ''' 计算三角形的面积 '''
    p = (a+b+c) / 2
    S = (p*(p-a)*(p-b)*(p-c))**(1/2)
    return S

line_lengths = [3,4,5]
area = triangle_area(line_lengths[0],line_lengths[1],line_lengths[2])
print(f"三角形的三边长分别为面积为{area}")
```

形参

形参列表中的变量名可以在函数体中使用

实参

p7_func_define.py

7. 函数 – 函数的参数传递

- 在Python中，从实参到形参的参数传递方式主要包括：位置参数、关键字参数、默认值参数、任意数目的位置参数、任意数目的关键字参数等。其中：
- 位置参数（positional argument）是基于实参的顺序进行的参数传递
- 关键字参数（keyword argument）是指将形参名称与实参的值进行关联的参数传递

```
def divide(numerator, denominator):
    ''' 两个实数相除 '''
    result = numerator / denominator # TODO 用try-except进行异常处理

# 方式1 位置参数 (positional argument)
print(divide(25, 100))
print(divide(100, 25))

# 方式2 关键字参数 (keyword argument)
print(divide(numerator=25, denominator=100))
```

p7_func_params-1.py

7. 函数 – 函数的参数传递

- 在函数定义时可以给形参指定默认值。
- 例如，定义函数计算一个圆柱的体积，形参包括底面半径、高度；若不指定高度，则高度为1：

```
def cylinder_volume(r, h=1):
    ''' 计算圆柱的体积 '''
    volume = 3.14159 * r**2 * h
    return volume

print(cylinder_volume(3))          # 未指定高度，高度为默认值1
print(cylinder_volume(3, 4))       # 指定高度，高度为4，覆盖掉默认值
```

- 需要注意的是，有默认值的参数必须排在所有无默认值参数的后面。
- 思考：为什么做出这样的语法规定？

```
def cylinder_volume(h=1, r):
    -snip-
```

左侧方式定义是否允许？为什么？

p7_func_params-2.py

(*) 7. 函数 – 函数的参数传递

- 当实参的个数不确定时，我们在定义函数时，可以使用形如 *args 接收传入任意数目的位置实参，此时，args可作列表处理；可以使用形如 **kwargs 接收传入任意数目的关键字实参，此时，kwargs可作字典处理（键为关键字名称，值为关键字对应的实参值）。
- 【任意数目位置参数】例如，定义函数接收任意数目的整数，计算其平方和：

```
def get_square_sum(*numbers):
    ''' 计算传入参数的平方和 '''
    the_sum = 0
    for number in numbers:
        the_sum += number**2
    return the_sum

print(get_square_sum(1, 2, 3))
print(get_square_sum(1, 2, 3, 4))
```

(*) 7. 函数 – 函数的参数传递

- 当实参的个数不确定时，我们在定义函数时，可以使用形如 *args 接收传入任意数目的位置实参，此时，args可作列表处理；可以使用形如 **kwargs 接收传入任意数目的关键字实参，此时，kwargs可作字典处理（键为关键字名称，值为关键字对应的实参值）。
- 【任意数目关键字参数】例如，定义函数接收任意数目关键字参数，构建包含用户信息的字典：

```
def build_profile(first, last, **user_info):
    profile = {'first_name': first, 'last_name': last}
    profile.update(user_info)
    return profile
```

将字典user_info的内容添加到字典profile
如果键已存在，用新的值覆盖原来的值

```
user = build_profile('Harry', 'Potter', location='Hogwarts', field='Wizarding')
print(user)
```

7. 函数 – 函数的参数传递机制：不可变对象的参数传递

- 在Python中，整数、浮点数、字符串、元组等类型均为不可变对象（immutable objects）。其本身不可改变，任何尝试修改该对象的操作都会导致创建一个新的对象。

```
x = 1000
print(f'Define `x=1000`, id(x) = {id(x)}')
x = x + 1
print(f'After `x=x+1`, id = {id(x)}')
```

id() 函数返回对象的唯一标识符，通常是其内存地址

- 若不可变对象作为参数传递给函数，即使在函数内部改变参数的值，原始变量也不会受到影响。例如：

```
def add_one(v):
    v = v + 1
x = 100
add_one(x)
print(x)
```

https://docs.python.org/3/glossary.html#term-immutable

p7_func_params-3.py

7. 函数 – 函数的参数传递机制：可变对象的参数传递

- 在Python中，列表、字典、集合等类型均为可变对象（mutable objects），其本身的改变不会新建变量，即其id值不会改变。

```
x = [1,1,2,3,5]
print(f'id(x) = {id(x)}')
x.append(8)    # 列表末尾新增一个元素
print(f'id(x) = {id(x)}')
```

- 若可变对象作为参数传递给函数，在函数内部对参数的改变会“映射”到原始变量。例如：

```
def make_things_zero(v):
    for i in range(len(v)):
        v[i] = 0
x = [1,1,2,3,5,8,13]
make_things_zero(x)
print(x)
```

p7_func_params-4.py

(*)7. 函数 – 函数的参数传递机制：传递列表参数

- 在Python中，列表类型均为可变对象（mutable objects），因此若函数实参为列表名，则函数体内对列表的任何修改都会“同步”到主函数。
- 但若传递的参数为列表切片，由于列表切片是原列表（一部分）的浅拷贝，因此函数体内对列表切片中的不可变对象的修改不会影响到主函数中的列表。
- 例如：

```
def make_things_zero(v):
    for i in range(len(v)):
        v[i] = 0

x1 = [1,1,2,3,5,8,13]
x2 = [21,34,55,89,144]

make_things_zero(x1)      # 传递列表名
make_things_zero(x2[:])   # 传递列表的切片（拷贝）
```

p7_func_params-5.py

7. 函数 – 函数的嵌套调用和递归调用

- 函数中可以调用其他函数（嵌套调用），或者调用自身（递归调用）。
- 例如，定义如下函数计算斐波那契数列的第n项：

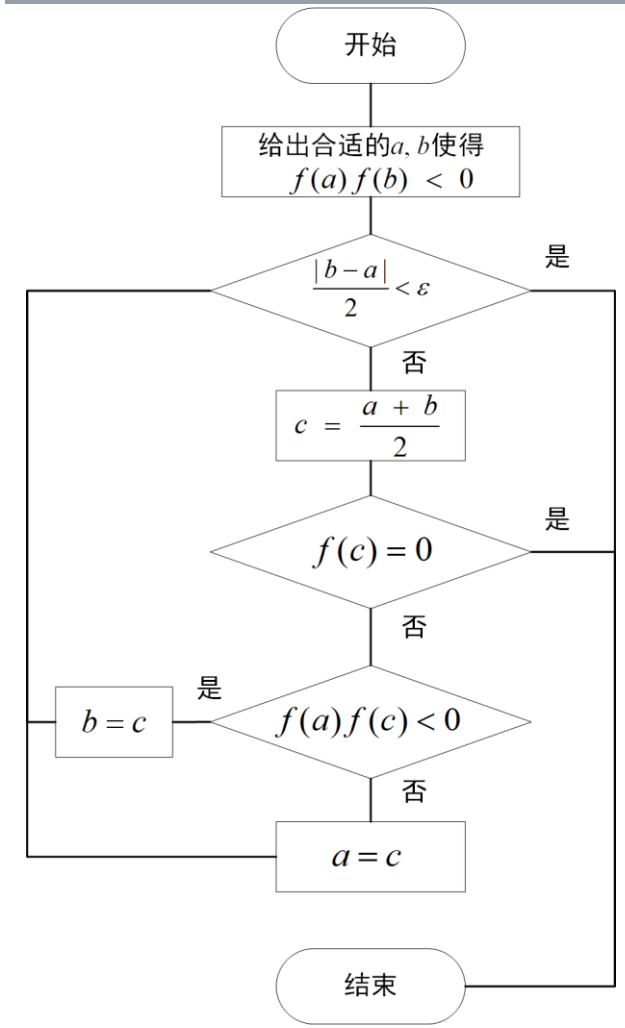
```
def Fibnacci(n):
    ''' 斐波那契数列的第n项(n >= 0) '''
    if n == 0 or n == 1:
        return 1
    else:
        return Fibnacci(n-1) + Fibnacci(n-2)
```

$$F_n = \begin{cases} 1 & (n = 0,1) \\ F_{n-1} + F_{n-2} & (n \geq 2) \end{cases}$$

- 数学上其他常用的递归关系：阶乘、组合数（杨辉三角形）、贝塞尔多项式、勒让德多项式、切比雪夫多项式，等等。

p7_func_recursive.py

7. 函数 – 接收函数对象为参数



二分法流程图

- 在Python中，函数也可以接收函数对象作为参数。
- 例如：定义如下函数实现通用的二分法数值求解非线性方程。

```
def bisection_method(a, b, f, tol=1e-8):
    -snip-
```

- 在实际调用时，可直接传入已定义的函数名。例如：

```
def func(x):
    return math.exp(x) + x**3 + math.sin(x)

root_bisection = bisection_method(-5, 0, func)
```

p7_func_params-6.py

7. 函数 – lambda 函数

- lambda函数是 Python 中的一种匿名函数，通常用于编写简短的、一行实现的函数。
- lambda函数的主体是一个单行表达式，该表达式的结果就是返回值。与def定义的函数不同，它不能包含多行逻辑。其基本语法为：

```
lambda 参数1, 参数2, ... : 表达式
```

- 例如：定义如下lambda函数计算平面上两点之间的距离，并将函数对象赋值给变量distance

```
distance = lambda x1,y1,x2,y2: ((x1-x2)**2 + (y1-y2)**2)**0.5

print(distance(0,0,3,4))
```

可以像调用一般的函数（通过def定义）
一样调用lambda函数

p7_func_lambda.py

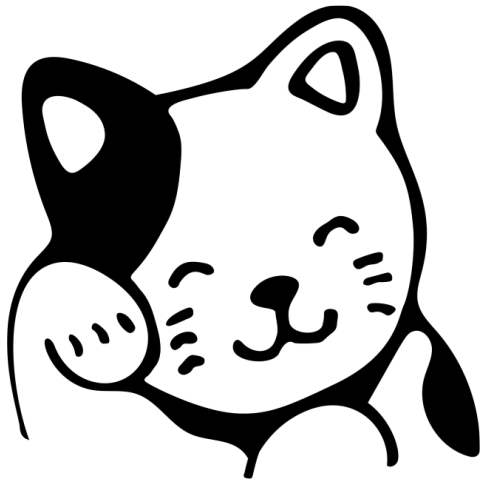
(*)7. 函数 – lambda 函数

- 之前定义的接收函数对象为参数的函数，也可以接收lambda函数对象为参数
- 在Python中，自带的map()，filter()，sorted()等函数都接收函数对象为参数

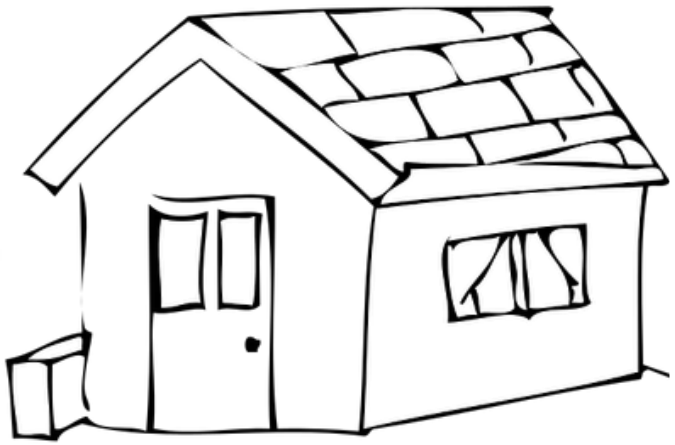
函数	语法	说明
map()	map(function, iterable) # function：接收一个参数	将一个函数应用到可迭代对象（如列表、元组等）的每一个元素上，并返回一个包含计算结果的迭代器
filter()	filter(function, iterable) # function：接收一个参数，并返回布尔值	依据某一条件过滤可迭代对象中的元素，并返回一个仅包含满足条件的元素迭代器
sorted()	sorted(iterable, key=None) # key：接收一个函数对象，用于从元素中提取用于排序的键	对可迭代对象进行排序，返回一个新的列表

8. 类和对象

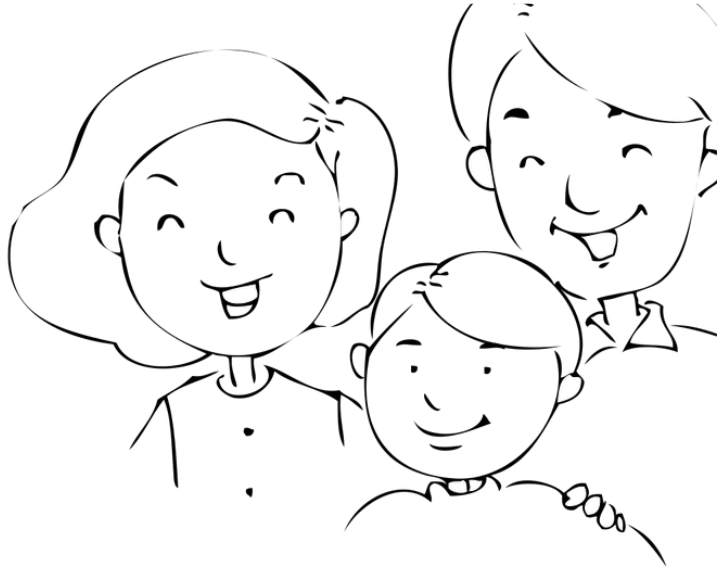
- 在Python中，可以采用面向对象编程（Object Oriented Programming, OOP）的思想编写程序。
- 在面向对象编程中，可定义表示现实世界事物和情境的类（class），并基于类创建对象（object）。



class Cat



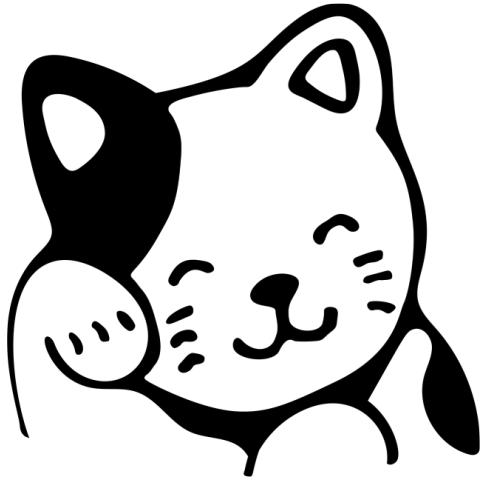
class House



class Human

8. 类和对象

- 在Python中，在定义类时，我们可以定义此类对象可能具备的通用属性和行为；当从类中创建具体对象时，每个对象都会自动具备这些通用属性和行为。



class Cat

Cat	
-name: str	名字
-age: int	年龄
-height: float	身高
-weight: float	体重
+meow(): None	喵喵叫
+catch_mouse(): None	捉老鼠
+eat(): None	吃饭

属性 ⟶ 成员变量

行为 ⟶ 成员函数（方法）

class diagram（类图）

p8_class.py

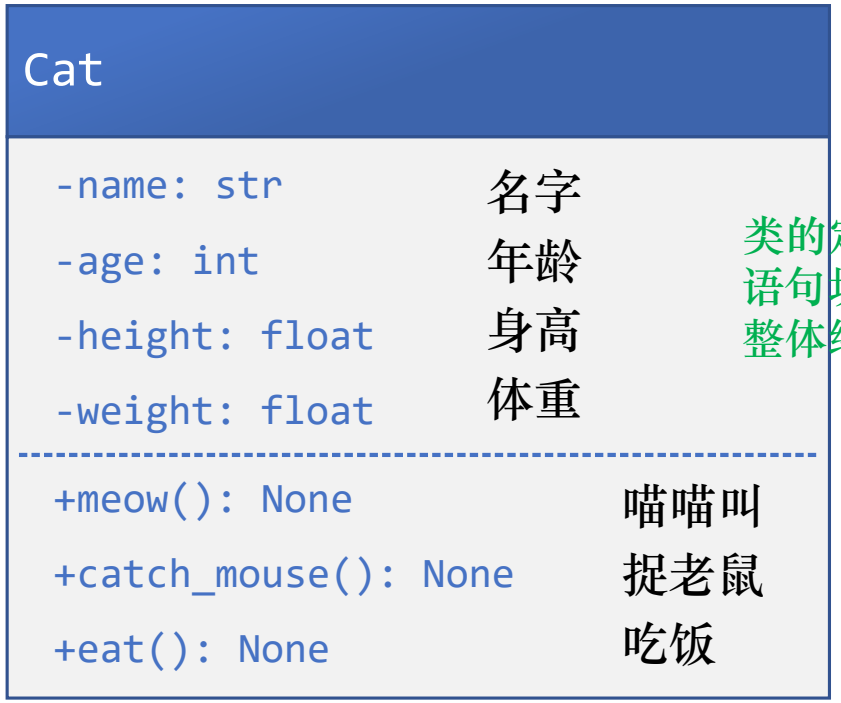
(*)8. 类和对象

- 从类创建对象的过程称为实例化（instantiation），相应的对象称为类的实例（instance）。
- 在Python中，可以用isinstance()判断对象是否是某个类的实例，可以用type()查看对象的类；整数、浮点数、字符串、布尔型、列表、字典等数据类型都是Python中自带的类。
- 例如：

```
my_cond = 2024 % 7 == 0

print('type(my_cond) =', type(my_cond))
print('isinstance(my_cond, bool) =', isinstance(my_cond, bool))
```

(*) 8. 类和对象

- 除此之外，用户可以自定义类，并用这些类创建对象。在Python中，通过class关键字定义类。
- 例如：定义一个Cat（猫）类，此类对象具备如下类图所示的属性和方法



类的定义中语句块需要整体缩进

```
class Cat:
    ''' 猫 '''
    def __init__(self,name,age,height,weight):
        self.name = name
        self.age = age
        self.height = height
        self.weight = weight
    def meow(self):
        -snip-
    def catch_mouse(self):
        -snip-
    def eat(self):
        -snip-
```

类的doc_string，对类的功能和方法进行说明

类的初始化函数，函数名固定，可在此函数中初始化类的成员变量

调用类的成员变量时，需加上self参数

类的成员函数（方法）

p8_class.py

(*) 8. 类和对象 – 类的实例化

- 在类的定义中，成员函数的定义和相关的语法规则与函数定义一致，但需注意：
- self参数在任何成员函数的参数列表中都是必需的，并且必须位于其他参数之前。
- self是对实例本身的引用，是每个独立的实例能够访问类中的属性和方法的必要条件；下面我们首先介绍类的实例化，例如，创建Cat类的一个实例cat，并用.访问类的成员变量与成员方法：

```
cat = Cat("小黑", 10, 30, 20)
```

参数传递取决于初始化函数（__init__()）的参数列表

cat对象的属性：

- name = "小黑"
- age = 10
- height = 30
- weight = 20

```
print(cat.name)
print(cat.age)
```

访问成员变量

```
cat.meow()
```

调用成员方法

p8_class.py

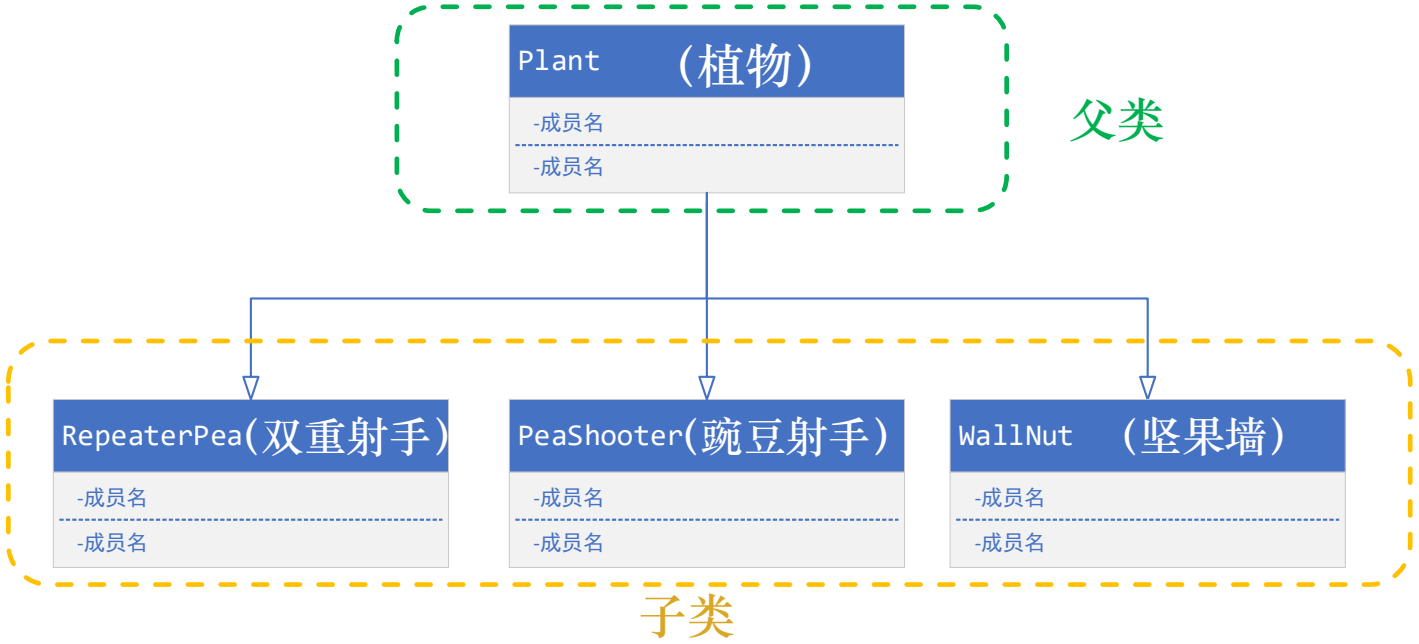
(*) 8. 类和对象 – 类的实例化

- 在安装过Python插件的VSCode编辑器中输入类的实例名，然后输入"."，会自动弹出包含类的成员方法、成员变量的提示框。例如：

(*) 8. 类和对象 – 类的继承

- 在Python中，可基于已有的类构建新的类，即类的继承（inheritance）。
- 一个类继承另一个类时，将自动获得另一个类的所有属性和方法，同时还可以定义自己的属性和方法，或重写（override）原有的类的属性或方法。原有的类称为父类（parent class）或超类（super class），而新类称为子类（child class或subclass）。

https://github.com/wszqkzqk/pypvz



9. 模块

- 在Python中，可以将众多功能相近或者具有关联的类、函数以及其他数据对象汇聚在一个Python文件中，作为一个模块（module）以供其他Python程序或模块调用。
- 模块名需满足Python变量命名规则：仅包括英文字母（或汉字）、数字、下划线，且不以数字开头。
- 例如，构建一个求解方程的根的模块，其中包含多种求解方法，适用于不同的情况：

```
''' 测试模块 本模块包含多个非线性方程求根函数 '''
import math

def quadratic(a, b, c):
    -snip-
def cubic(a, b, c, d):
    -snip-
def bisection_method(f, a, b, tol=1e-8, max_iter=100):
    -snip-
```

模块的doc_string，对模块的功能进行简要说明

root_finding.py

p9_module_example

9. 模块

- 可以使用import语句导入模块，通过“.”引用其中的变量、函数或类
- 方法1：import 模块名称（不带文件后缀“.py”）

```
import root_finding
results = root_finding.cubic(1,0,0,-1)
```

- 方法2：import 模块名称 as 别名（可以起到简化书写的作用）

```
import root_finding as rf
results = rf.cubic(1,0,0,-1)
```

- 方法3：from 模块名称 import 模块中的变量或函数或类 as 别名（可以只引入部分变量、函数或类）

```
from root_finding import bisection_method as bisect
result = bisect(lambda x:(-1+2*x**4+x**3), 0, 1)
```

p9_module_example

9. 模块 – Python第三方库

- 除了自带的标准库，Python还拥有众多的第三方库，涵盖科学计算、数据分析与可视化、Web开发、办公应用、机器学习等多个领域，可以有效节省开发时间，避免重复“造轮子”。大多数第三方库都有活跃的社区支持和文档，便于用户学习和使用。
- 第三方库可以通过Python自带的包管理工具pip进行安装和管理。

命令	功能
`pip install <package_name>`	安装第三方库
`pip uninstall <package_name>`	卸载第三方库
`pip list`	查看已安装的库
`pip install -r requirements.txt`	安装文件中的所有库

pip镜像源配置可参考：https://mirror.tuna.tsinghua.edu.cn/help/pypi

9. 模块 – Python第三方库 – jupyter notebook介绍

- Jupyter Notebook 是一种交互式计算平台，能够逐个单元格执行代码并查看即时输出，适合应用于实验性分析和教学演示。
- Jupyter Notebook 支持 Python 在内的多种编程语言，同时可以方便地结合 Markdown 语法，能够将代码、文本、公式和可视化图表集成在一个文档中，便于记录和共享研究过程，在数据分析、机器学习和科学计算等领域有广泛应用。
- 安装jupyter模块：

```
pip install jupyter
```

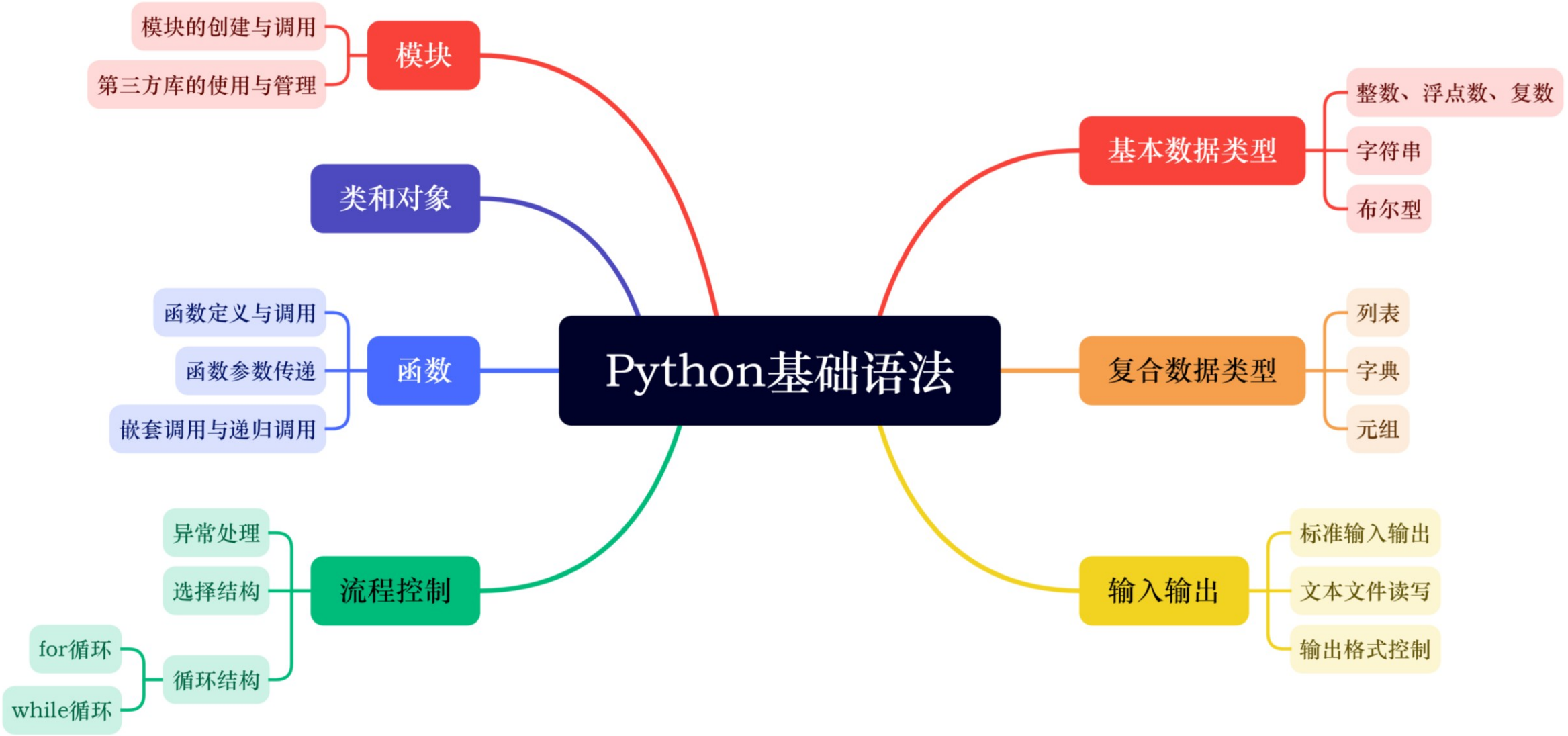
- 使用Jupyter Notebook：
  - 方法1：在终端输入命令：

```
jupyter notebook
```

  - 方法2：VSCode + 扩展插件

jupyter_notebook_example

本节小结


Python基础语法
模块
模块的创建与调用
第三方库的使用与管理
类和对象
函数
函数定义与调用
函数参数传递
嵌套调用与递归调用
流程控制
异常处理
选择结构
循环结构
for循环
while循环
基本数据类型
整数、浮点数、复数
字符串
布尔型
复合数据类型
列表
字典
元组
输入输出
标准输入输出
文本文件读写
输出格式控制

清華大學
Tsinghua University

感谢倾听!

孙鑫礼

2025年10月26日

课后反馈

反馈问卷

乐学公众号

Reference

- Matthes E. Python Crash Course, 2nd Edition[M]. San Francisco: No Starch Press, Inc., 2019.
- Some icons or logos are from the website of https://icons8.com
- Some images are from the website of https://pixabay.com
- Some schematics are drawn by Xmind or Visio