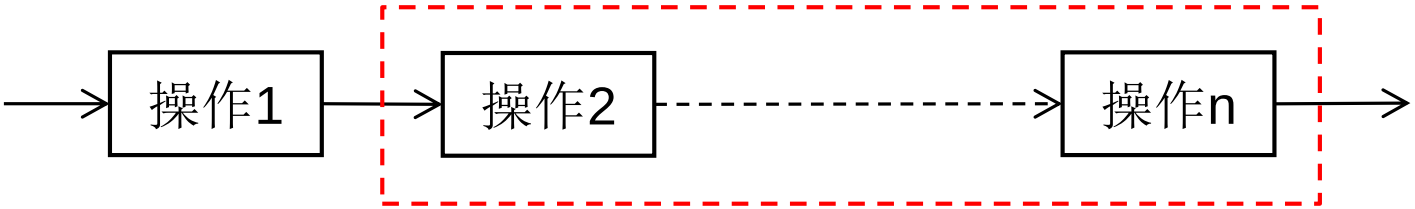
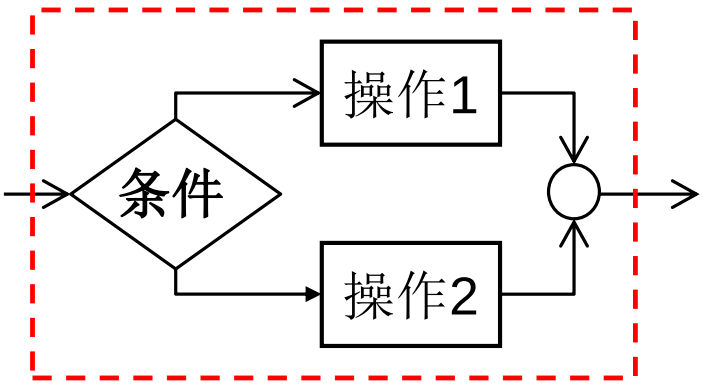
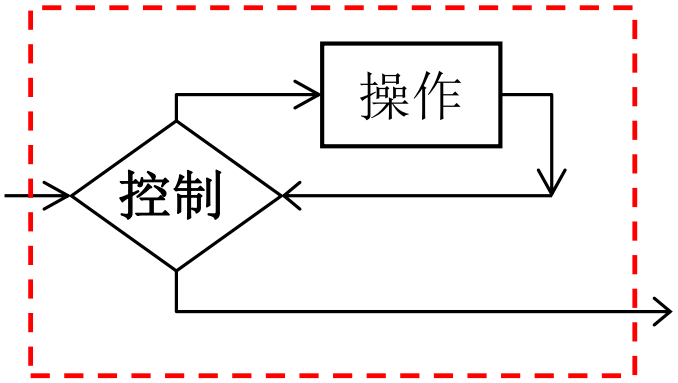
操作、控制和抽象

- 顺序执行模式：用语句组描述



- 选择和重复执行模式：用 if 语句和循环结构描述

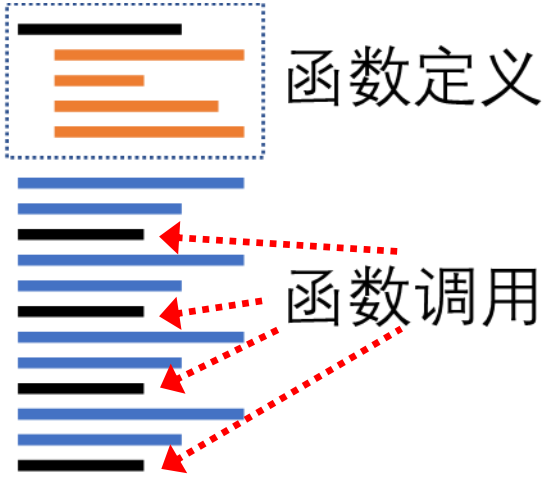




- 一段计算过程也可以抽象地看作一个 (组合) 操作
  - 有利于分解复杂的程序和计算

计算的抽象 — 函数

- 需求：包装一个计算过程；重复使用一个代码片段 (一段计算)
- 代码片段的抽象：将一段代码定义为一个计算体并命名 — 定义函数抽象
  - 一个函数定义 (define) 描述某种特定计算过程
  - 定义好的函数可以通过函数名字，以简单的方式 、任意多次被调用
    - 一个函数被调用 (call)，即是在给定一组具体实参的情况下，完成一次相应的计算过程
- Python 允许自己定义函数 — 自定义函数
  - 函数定义：包装一段代码，建立一个函数抽象，并为其命名



定义函数 — def 语句

- 语法：

```
def 函数名 ([参数列表]) :          # 函数头部
    语句组                         # 函数体
```

  - 参数列表：以逗号分隔的 0 个或多个名字，称为形式参数 (parameter，简称为形参)

- 语义：函数定义被看作一种复合语句；一个函数定义被执行时，解释器根据形式参数表和函数体建立一个函数对象，并将其约束到给定的函数名

  - 函数体里的 return 语句用来描述函数的返回值

- 自定义函数的使用：通过函数调用的方式

  - 执行一个函数调用 (表达式)，即是在给定实参 (argument) 的情况下执行其函数体所包装的语句，完成一次计算

函数的定义和调用

- **return** 语句只能用在函数定义里，有两种形式：

  ① **return** 表达式

  ② **return**

  - 语义：执行 **return** 语句导致当前函数执行结束；如含有表达式，就以表达式的值作为函数返回值；否则函数返回 **None**

- 解释器执行一个函数调用表达式：

  1. 根据被调用函数名找到所执行的函数对象
  2. 从左到右逐个求值实参 (表达式)，并把实参值和对应的形参相约束，然后执行函数体里的语句
  3. 遇 **return** 语句执行结束，求值其表达式作为函数返回值，没有表达式时返回 **None**
  4. 执行完函数体所有语句函数也结束，返回 **None**

定义函数

- 例：计算三角形面积函数的改进和变形
- 例：计算并输出三角形面积的函数 (定义无返回值的函数)
- 例：阶乘函数和交互式阶乘计算器
- 函数定义，是建立计算抽象 (过程抽象) 的机制
  - 定义好的函数可以很方便地调用
  - 一次定义，任意多次使用
  - 把任意复杂的一段段程序代码抽象为一个个函数，有利于把复杂的程序按层次结构进行分解
  - 定义包含许多函数的模块，可供以后使用；有价值的模块还可以提供给别人使用

一些 Python 机制 (1)

- 允许在一行里写几个语句
  - 形式：语句之间用分号分隔
  - 语义：按顺序逐个执行这些语句
  - 语用：可以将几个短语句写在一行，缩短代码的行数 (实际并不提倡)；Python 的习惯是一行只写一条语句
- 条件表达式：带条件的表达式，计算得到一个值
  - 语法：

    表达式1 if 条件 else 表达式2　　*# 例：* x if x >= 0 else -x
  - 语义：先求值条件，如果条件为真就以 表达式1 的值作为整个表达式的值，否则就以 表达式2 的值作为整个表达式的值
  - 注意：条件表达式与条件语句的不同！语句没有值，执行时产生“效果”，表达式的计算就是求值 (例：绝对值函数)

一些 Python 机制 (2)

- 函数的文档串：
  - 在函数定义的 **def** 子句之后 **(即函数体的首句)**，建议写一个“三引号括号”形式的长字符串，简短地描述函数的用途、使用方式、副作用等
  - **Python** 的内置函数，标准库里的函数等都有文档串，可通过 函数名.__doc__ 来查看

```
>>> print(pow.__doc__)
Equivalent to x**y (with two arguments) or x**y % z (with three arguments)

Some types, such as ints, are able to use a more efficient algorithm when
invoked using the three argument form.
```

一些 Python 机制 (3)

- 检查数据类型
  1. **type(表达式) == 类型名**
  2. 标准做法： **isinstance(表达式, 类型名)**
     - 检查表达式的值 (对象) 是否属于某个类型或者其 (直接或者间接的) 子类型
     - 所有类型都是内置类型 **object** 的 (直接或间接) 子类型

```
>>> type(2.3) == float
True
>>> type(2.3) == int
False
```

```
>>> type(2.3) == object
False
```

```
>>> isinstance(2.3, float)
True
>>> isinstance(2.3, int)
False
>>> isinstance(2.3, object)
True
```

一些 Python 机制 (4)

- **assert 断言语句**

  assert 条件　　# *条件为逻辑值表达式，称为断言*

  assert 条件, 表达式　　# *表达式可以为任意表达式*

  - 语义：求值条件，当其值为真时，程序继续执行；否则，报 **AssertionError** 错误，并导致程序的执行终止
    - 对第二种形式，当条件为假时，则继续求表达式的值，并将其作为 **AssertionError** 的参数送出
  - 语用：只应该用来描述程序 / 函数正确执行的必要条件，辅助程序调试 **(不能当成条件语句使用)**

Python 解释器的工作方式

- Python 系统 (解释器) 按行读入和处理
  - 默认一个语句 (或表达式) 中间不能换行
  - 如果一行中存在未配对的括号，则认为下一行是本行的继续
  - 可以在一行最后用续行符 \ 说明下一行是本行的继续

示例：语句/表达式不能跨越多行，将报错

```
if a > 0 and b > 0 and c > 0 and a+b > c
   and a+c > b and b+c > a:
```

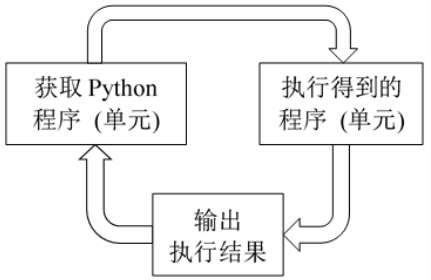
示例：括号不匹配时自动延伸

```
if (a > 0 and b > 0 and c > 0 and a+b > c
    and a+c > b and b+c > a) :
```

示例：使用续行符号 \

```
if a > 0 and b > 0 and c > 0 and a+b > c\
   and a+c > b and b+c > a :
```

Python 解释器的工作方式



Python 解释器的工作循环

- Python 系统 (解释器) 按行读入和处理
  - 默认一个语句 (或表达式) 中间不能换行
  - 如果一行中存在未配对的括号，则认为下一行是本行的继续
  - 可以在一行最后用续行符 \ 说明下一行是本行的继续
  - 如所读行是某复杂结构的头部，会根据代码的退格形式继续读完整个结构，之后再处理

  程序中的一条逻辑行可能由一条或几条物理行构成
- Python 执行环境的基本循环：
  - 读入一个逻辑程序行，或者一个跨几行的复杂结构，如 **def, for, while** 等
  - 处理读入的结构完成操作，输出计算结果 (或者报错)
- 处理脚本中代码的规则也一样，只是不显示求值结果