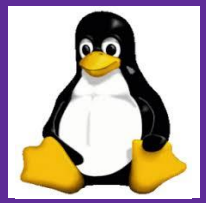
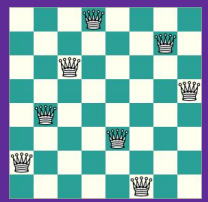
Introduction to Computation

Autumn, 2024

Prof. Hongfei Fu
Shanghai Jiao Tong University
(slides by Prof. Fang Cheng)

fuhf@cs.sjtu.edu.cn
https://jhc.sjtu.edu.cn/~hongfeifu/
https://github.com/ichengfan/itc





4

Outline

- Conditional Statement
- Loop Statement
- Function

Conditional Statement

Boolean value

Python has two Boolean values: True, False

- True and False are bool type
- True is 1 and False is 0. Both are bool type
- bool(x): returns the Boolean value of a specified object
  - Any nonzero number is interpreted as “True”
  - Any nonempty string is “True”
  - Return False for [], (), {}, False, 0, None
- In logic expression, x is interpreted as bool(x)
- bool is a special type of int
- ==
  - True is 1: $123 \neq 1$
  - None is not the same as 0, False, or an empty string

```
23  print(type(True), type(False))
24  print(True == 1, False == 0)
25
26  print(bool(123), bool("123"))
27  print(bool(0), bool(""))
28  print(bool(None))
29
30  print(123 == True, "123" == True)
31  print(None == False)
```

```
<class 'bool'> <class 'bool'>
True True
True True
False False
False
False False
False
```

Boolean expression

A Boolean expression is an expression that is either true or false

- Comparison Operators that can create Boolean expression
  - >, <, >=, <=, ==, !=, in, not in (集合)
  - is, is not (will discuss it later)
  - all comparison operations in Python have the same priority

```
print(1+2+3>6 and "abc">"ABC")
print(1+2+3>6 or "abc">"ABC")
print(not (1+2+3>6))
print(not ("abc">"ABC"))
```

```
False
True
True
False
```

- There are three logical operators: **and**, **or**, and **not**
  - x and y returns True, if and only if both x and y are True
  - x or y returns True, if and only if at least one of them are True
  - not x returns True, if and only if x is False
  - The priority: **not > and > or**. Use "()" if you are not sure

```
print(5 and "abc">"ABC")
print(0 and "abc">"ABC")
print(1+2+3>6 or 3)
print(not 3.1)
```

```
True
0
3
False
```

- Short-circuit evaluation: In some situations, knowing one input to an operator is enough to determine its value. In those cases, the other input is not evaluated.
  - True or x, False and y. x and y will not be evaluated
- A Boolean expression may not directly return True or False, but some equivalent values
  - 1+2+3>6 or 3, return 3, bool(3)==True
  - 0 and "abc">"ABC", return 0, bool(0) ==False

在λ表达式有巧妙的用处

Chained Comparison

Comparisons can be chained arbitrarily.

- For example, x < y <= z is equivalent to x < y and y <= z, except that y is evaluated only once (but in both cases z is not evaluated at all when x < y is found to be false)
- Formally, if a, b, c, …, y, z are expressions and op1, op2, …, opN are comparison operators, then an op1 b op2 c … y opN z is equivalent to a op1 b and b op2 c and … y opN z, except that each expression is evaluated at most once
- 严禁链式赋值语句：**x=y=1**

```
1  def test_comp_chain(x, y, z):
2      print(x < y < z, x < y and y < z)
3      print(x < y > z, x < y and y > z)
4      print(x > y < z, x > y and y < z)
5      print(x > y < z, x > y and y < z)
6
7
8  test_comp_chain(1, 1, 1)
9  test_comp_chain(1, 2, 3)
10 test_comp_chain(1, 3, 2)
11 test_comp_chain(3, 2, 1)
```

```
False False
False False
False False
False False
True True
False False
False False
False False
False False
True True
False False
False False
False False
False False
False False
False False
```

和数学中一样

https://docs.python.org/3/library/stdtypes.html

Conditional statement: if

```
36  x = 100
37
38  if x > 0:
39      print("Hello world")
40
41  if x <= 100:
42      print("Good")
43
44  if x > 1000:
45      print("Nothing")
46
47  print("End")
```

```
Hello world
Good
End
```

- A conditional statement is:

  if <Boolean expression>: # header
  FIRST STATEMENT # body
  ...
  LAST STATEMENT

- The Boolean expression after the if statement is called the **condition**.
  - If it is **True**, then the indented statement gets executed.
  - If not, nothing happens
- The if statement is made up of a header and a block of statements
  - The header begins on a new line and ends with a colon (:). The indented statements that follow are called a block. A statement block inside a compound statement is called the body of the statement
- There is no limit on the number of statements that can appear in the body of an if statement, but there has to **be at least one**. You could use **pass** to skip this issue

```
x = 100

if x>100:
    pass # do nothing

print("End of the program.")
```

```
End of the program.
```

Indent 缩进

```
x = 100
if x > 7:
    print("Hello world")
    print("Monday")
```

```
Hello world
Monday
```

```
x = 1
if x > 7:
    print("Hello world")

print("Monday")
```

```
Monday
```

```
x = 1
if x > 7:
    print("Hello world")
        print("Monday")


print("Monday")
```

缩进具有严格的要求，不可以随便添加
连续的相同缩进的语句构成一个逻辑上的整体

if: example

- Check whether an integer is even or not
- Check whether a year is a leap year (闰年)
- Check whether your score is above 60.0 to pass the exam

```
49  n = 2018
50
51  if n%2 == 0:
52      print("An even integer")
53
54  y = 2014
55  if (y%4 == 0 and y%100 != 0) or (y%400 == 0):
56      print("A leap year")
```

```
score = 80.1
if score >= 60:
    print("Passed")
```

if else

- There are two possibilities, and the condition determines which one gets executed.

```
if <Boolean expression>:
    do_sth_here #
else:
    do_sth_else_here
```

如果xxx发生，……
否则，……

- If the Boolean expression is True, the statement block after “if:” will be executed.
- Otherwise, the statement block after “else:” will be executed.

```
58  n = 2011
59  if n%2 == 0:
60      print("I will get up early")
61  else:
62      print("I will go to library")
```

```
64  n = -3
65
66  if n >= 0:
67      x = n + 1
68  else:
69      x = 3*n + 1
70
71  print(x)
```

Exactly one branch will be executed

Chained conditionals

- In practice, we need to consider more complicated conditions:

```
if <Boolean expression 1>:
    do_sth_here_1
elif <Boolean expression 2>:
    do_sth_here_2
        ......
elif <Boolean expression n>:
    do_sth_here_n
else:
    do_else_here
```

```
if x == 1:
    print("Monday")
elif x == 2:
    print("Tuesday")
elif x == 3:
    print("Wednesday")
elif x == 4:
    print("Thursday")
elif x == 5:
    print("Friday")
elif x == 6:
    print("Saturday")
else:
    print("Sunday")
```

- The program checks the Boolean expression from 1 to n, if one of them is satisfied, then the statement block will be executed. Or the “else:” statement block is executed
- elif is an abbreviation of “else if.” Again, exactly one branch will be executed
- There is no limit of the number of elif statements
- else is optional (有些语言必须有，python可选)

If…elif…else: example

- (3n+1 problem) Given a positive integer n, if n is even, then n = n/2; if n is odd, n = 3n+1. Will n =1 after a finite number of steps?

```
n  = 300

if n == 1:
    print("1 is reached.")
elif n %2 == 0:
    n //= 2
else:
    n = 3*n +1

print(n)
```

The program will check n ==1 first, then n%2==0. If none is satisfied, it will execute “else:”

- (piecewise function) $f(x) = \begin{cases} x^3, & x < -1 \\ x, & -1 \le x < 1 \\ x^2, & x > 1 \end{cases}$

```
def f(x):
    if x<-1:
        return x ** 3
    elif x>=-1 and x < 1:
        return x
    else:
        return x**2

print(f(-3))
print(f(0.5))
print(f(7))
```

```
-27
0.5
49
```

When you write a conditional expression, you should check your conditions. Don’t miss any conditions! Or some code will never be executed

Nested expressions

- Expressions can be nested (嵌套)
- Indentation

```python
def select(n):
    if n % 4 == 0:
        if n % 3 == 1:
            print(f"{n} % 12 == 4")
        elif n % 3 == 2:
            print(f"{n} % 12 == 8")
        else:
            print(f"{n} % 12 == 0")
    elif n % 4 == 1:
        if n % 3 == 1:
            print(f"{n} % 12 == 1")
        elif n % 3 == 2:
            print(f"{n} % 12 == 6")
        else:
            print(f"{n} % 12 == 9")
    else:
        print("End")


select(0)
select(1)
select(4)
select(6)
select(8)
select(9)
```

```
0 % 12 == 0
1 % 12 == 1
4 % 12 == 4
End
8 % 12 == 8
9 % 12 == 9
```

Structural Pattern Matching (3.10)

- match subject:
  case <pattern_1>:
    <action_1>
  case <pattern_2>:
    <action_2>
  case <pattern_3>:
    <action_3>
  case _:
    <action_wildcard>
- 从上往下对比，如果遇到符合的pattern，就执行该case的代码，结束后离开match
- _ : 表示wildcard，会match任意的情况，可以省略
- 比C/C++中switch要强很多

```
if x == 1:
    print("Monday")
elif x == 2:
    print("Tuesday")
elif x == 3:
    print("Wednesday")
elif x == 4:
    print("Thursday")
elif x == 5:
    print("Friday")
elif x == 6:
    print("Saturday")
else:
    print("Sunday")
```

```
1  def test_day(x):
2      match x:
3          case 1:
4              print("Monday")
5          case 2:
6              print("Tuesday")
7          case 3:
8              print("Wednesday")
9          case 4:
10             print("Thursday")
11         case 5:
12             print("Friday")
13         case 6:
14             print("Saturday")
15         case _:  # try to remove it.
16             print("Sunday")
17
18
19 for x in range(1, 10):
20     test_day(x)
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Sunday
Sunday
```

结构更清晰，可读性更好

Tips

- 注意细节

```python
def f(n):
    if n == 1:
        return True
    else:
        return False


# 等价于
def f(n):
    return n == 1
```

```python
def f(n):
    if g(n):
        return a(n)
    else:
        return b(n)


# 等价于
def f(n):
    return a(n) if g(n) else b(n)
```

```python
# Bad
if v == True:
    f()

# Good
if v:
    f()

# Bad
if v == False:
    f()

# Good
if not v:
    f()
```

Loop Statement

Loop

问题：如何重复输出”hello world”, 3次
答案：反复调用print三次
问题：如果要输出n次呢？n是一个变量
答案：……

《CS版从三到万》

1. while statement
2. for statement

从前有座山，山上有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，故事讲的是从前有座山，山上有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，故事讲的是从前有座山，山上有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，故事讲的是从前有座山，山上有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，故事讲的是从前有座山，山上有座庙，庙里有个老和尚，老和尚在给小和尚讲故事，故事讲的是。。。

while

- The grammar of while statement:

  while <Boolean expression>:

  do_sth_here #

When the Boolean expression is True, the statement block will not stop executing

- Examples:
  - Repeats printing “Get up early” three times
  - Compute $\sum_{i=1}^{100} i$
  - Given n, compute 3n+1
  - Given an integer n, compute the number of “0” at the end of n

```
i = 0
while i<3:
    print("Get up early tomorrow")
    i += 1
print(i)
```

```
1  total = 0
2  i = 1
3
4  while i <= 100:
5      total += i
6      i += 1
7
8  print(f"Sum = {total}")
```

```
1  n = 300
2  print(f"n is {n}")
3
4  while n != 1:
5      if n % 2 == 0:
6          n //= 2
7      else:
8          n = 3 * n + 1
9
10 print(f"End n = {n}")
```

The Boolean expressions are not satisfied when i =3, i =101 and n = 1

for

- In mathematics, $x \in A$. In python, we say x in A. in is a keyword in python
  - $x \notin A$: x not in A. (not is also a keyword)
- When we have a collection of elements, we can use **for** to construct a loop:

  for x **in** collection:　　（自动枚举合集中的每个元素）

  　　do_sth
  - Here the collection can be type of **list, dict, tuple, set, str, range**, etc.
- For all x ∈ collection, the statement block (do_sth) will be repeatedly executed

```
str1 = "CatEatMouse"
for s in str1:
    print(s, end=" ")
print()

for x in range(5):
    print(x , end=" ")
print()
```

```
C a t E a t M o u s e
0 1 2 3 4
```

```
zoo = ["dog", "cat", "mouse", "dragon", "tiger"]

for animal in zoo:
    print(animal)

zoo_number={"dog":1, "cat":3, "mouse":5, "dragon":0, "tiger":7}

for animal in zoo_number.keys():
    print(zoo_number[animal])
```

```
dog
cat
mouse
dragon
tiger
1
3
5
0
7
```

可迭代的(iterable): list, dict, tuple, set, str, range

range()

- range(n): generate a collection of integers from 0 to n −1.
- range(a, b, d): generate a collection of integers from a to b, with step d .
  - (从a到b以d为公差的等差数列)
  - a, b, d必须都为整数
- for x in range(a, b, d):
  do_sth

```
for x in range(10):
    print(x, end=" ")
print()

for x in range(1, 10, 1):
    print(x, end=" ")
print()

for x in range(1, 10, 2):
    print(x, end=" ")
print()

for x in range(1, 10, 3):
    print(x, end=" ")
print()

for x in range(10, 1, -2):
    print(x, end=" ")
print()
```

，可以省略.

```
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 3 5 7 9
1 4 7
10 8 6 4 2
```

```
1  x = range(100)
2  print(x, type(x), len(x))
```

range(0, 100) <class 'range'> 100

```
num_sum = 0
for x in range(101):
    num_sum += x

print(num_sum)
```

```
5050
```

for VS. while

- for 自动枚举一个数据中的每个元素
- while需要自己枚举每个情况

```
16  total = 0
17  i = 1
18  while i <= 100: # 只要i不大于100
19      print(i)
20      total += i
21      i += 1
22
23  print(total)
```

```
26  total = 0
27
28  for i in range(100): # i会自动枚举0-100的情况
29      print(i)
30      total += i
31
32  print(total)
```

```
for x in range(5):
    pass
print(x)

x = 0
while x<5:
    x += 1
print(x)

x = 0
while x in range(5):
    x += 1
print(x)
```

```
4
5
5
```

for x in xxx 结束时，x时xxx中最后一个元素
while x in range(n): 结束时，x=n

break and continue

break, continue: quit the current iteration of the loop statement

- break: The break statement is used to immediately leave the body of its loop.
- continue: This is a control flow statement that causes the program to immediately skip the processing of the rest of the body of the loop, for the current iteration. But the loop still carries on running for its remaining iterations

```
for i in [12, 16, 17, 24, 29]:
    if i%2 == 1:
        break
    print(i)
print("Break and done")

for i in [12, 16, 17, 24, 29]:
    if i%2 == 1:
        continue
    print(i)
print("Continue and done")
```

```
12
16
Break and done
12
16
24
Continue and done
```

Break and continue will quit the current iteration.
However, continue will execute the next iteration. (再来一轮)
Break won't but quit the whole loop. (认输投降)

break, continue, return

- break: 结束整个循环
- continue：结束本轮循环，开始下一轮
- return: 结束函数运行，返回（可以不带返回值）

```
1298    for i in range(7):
1299        if i % 3 == 2:
1300            print(" ", end=" ")
1301            return #error, must in a function
1302        print(i, end=" ")
1303    print("\nTest return done.")
```

```
1284    for i in range(7):
1285        if i % 3 == 2:
1286            print(" ", end=" ")
1287            break
1288        print(i, end=" ")
1289    print("\nTest break done.")
1290
1291    for i in range(7):
1292        if i % 3 == 2:
1293            print(" ", end=" ")
1294            continue
1295        print(i, end=" ")
1296    print("\nTest continue done.")
```

```
0 1
Test break done.
0 1   3 4   6
Test continue done.
```

```
  File "C:\Users\popeC\OneDrive\CS124计算导论\2023 秋季\course_code.py", line 1301
    return #error, must in a function
    ^^^^^^
SyntaxError: 'return' outside function
```

```
1305    def test_return():
1306        for i in range(7):
1307            if i % 3 == 2:
1308                print(" ", end=" ")
1309                return
1310            print(i, end=" ")
1311        print("\nTest return done.")
```

```
0 1
```

编程规范：for + for不超过两层。
特别是有return, break的情况下。
超过两层，第三层的逻辑用函数实现

return … if … else …

return value1 if condition else value2

- A convenient usage of return
- Equivalent

```
if condition:
    return value1
else:
    return valuel2
```

```python
def f(n):
    if n % 2 == 0:
        return n // 2
    else:
        return 3 * n + 1


def test_return_else(n):
    return n // 2 if n % 2 == 0 else 3 * n + 1


for x in range(10):
    print(f(x) == test_return_else(x))
```

```
True
True
True
True
True
True
True
True
True
True
```

```python
x = 12

y = x // 2 if x % 2 == 0 else 3 * x + 1

print(y)
```

for/while - else

- for/while loops also have an else clause which most of us are unfamiliar with
- The else clause executes after the loop completes normally
  - This means that the loop did not encounter a break statement.

```python
def test_for(n):
    print("n = {}:".format(n))

    for i in range(6):
        print(i)
        if i == n:
            break
    else:
        print("i is no longer less than 5")

    print(i)

test_for(2)
test_for(4)
test_for(6)
test_for(7)
test_for(8)
```

```
n = 2:
0
1
2
2
n = 4:
0
1
2
3
4
4
n = 6:
0
1
2
3
4
5
i is no longer less than 5
5
```

```
n = 7:
0
1
2
3
4
5
i is no longer less than 5
5
n = 8:
0
1
2
3
4
5
i is no longer less than 5
5
```

如果不用else：
在循环结束后加一个判断
for要注意，结束的情况和while不同

```python
def test_while(n):
    print("n = {}:".format(n))
    i = 1

    while i < 6:
        print(i)
        i += 1
        if i == n:
            break
    else:
        print("i is no longer less than 6")

    print(i)

test_while(2)
test_while(4)
test_while(6)
test_while(7)
test_while(8)
```

```
n = 2:
1
2
n = 4:
1
2
3
4
n = 6:
1
2
3
4
5
6
```

```
n = 7:
1
2
3
4
5
i is no longer less than 6
6
n = 8:
1
2
3
4
5
i is no longer less than 6
6
```

如果不用else：
在循环结束后加一个判断：if i==6

for/while – else 用例

- 在实际中，我们经常需要遍历一个数据结构，看看是否存在不符合要求的元素
- 常规的写法，使用一个flag变量，初始化为True。如果遇到不符合要求的元素，设置为False，然后退出。循环结束后，根据flag的值可以判断具体的情况
- 现在用 for/while –else, 可以不用flag
- 如果没有break语句，那么else永远都会执行

```
1  def is_valid(n):
2      return 0 < n < 10000
3
4
5  flag = True
6  for i in range(1, 1000):
7      if not is_valid(i):
8          flag = False
9          break
```

```
1  for i in range(1, 1000):
2      if not is_valid(i):
3          break
4  else:
5      print("All the elements are valid.")
```

All the elements are valid.
All the elements are valid.

Bit operator

- In Python, bitwise operators are used to perform bitwise calculations on integers
- The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators
- Then the result is returned in decimal format

&	Bitwise AND	x & y	Bitwise AND operator: Returns 1 if both the bits are 1 else 0.
|	Bitwise OR	x | y	Returns 1 if either of the bit is 1 else 0
~	Bitwise NOT	~x	Returns one's complement of the number ($0 \rightarrow 1,\ 1 \rightarrow 0$)
^	Bitwise XOR	x ^ y	Returns 1 if one of the bit is 1 and other is 0 else returns 0
>>	Bitwise right shift	x>>	Shifts the bits of the number to the right and fills 0 on voids left as a result. Similar effect as of dividing the number with some power of two
<<	Bitwise left shift	x<<	Shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two

Bit operator

x	y	x&y	x|y	~x	x^y	>>	<<
0	0	0	0	1	0	0	0
0	1	0	1	1	1	0	0
1	0	0	1	0	1	0	2
1	1	1	1	0	0	0	2

1111>>1, 111 1110>>1, 111
1111<<1, 11110, 1110<<1, 11100

$$\sim x := -(x+1)$$

The bitwise inversion of x is defined as $-(x+1)$

$$x \gg n := x//2^n$$

$$x \ll n := x * 2^n$$

Precedence: Bitwise operators is lower than arithmetic operators 1+n&2 is (1+n)&2

```python
num1 = 0b11010011
num2 = 0b11011100

print(num1, num2)
print(num1 & num2, num1 | num2)
print(~num1, ~num2)
print(num1 ^ num2)
print(num1 >> 1, num2 >> 1)
print(num1 << 1, num2 << 1)

print(f"{num1 : b} {num2 : b}")
print(f"{num1 & num2 : b}", f"{num1 | num2 : b}")
print(f"{~num1 : b}", f"{~num2 : b}")
print(f"{num1 ^ num2 : b}")
print(f"{num1 >> 1 : b}", f"{num2 >> 1 : b}")
print(f"{num1 << 1 : b}", f"{num2 << 1 :b}")
```

```
211 220
208 223
-212 -221
15
105 110
422 440
 11010011  11011100
 11010000  11011111
-11010100 -11011101
 1111
 1101001  1101110
 110100110 110111000
```

Bitwise operator in logic expression

```
1  n = 127
2  while n%2 == 1:
3      n //= 2
4  print(n)
5
6  n = 127
7  while n%2:
8      n //= 2
9  print(n)
10
11 n = 127
12 while n&1:
13     n >>= 1
14 print(n)
```

```
0
0
0
```

- 三种写法：Line 1-4 正常写法
- 第7行， n%2不为0的时候，永远为True
- 第12行，n&1不为0的时候，永远为True；n&1=0, n为偶数
- 第13行，n>>1 等价于 n//2

- [], (), {}, False, 0, None都为False
- 尽量采用第一种写法，可读性好。现在的编译器很智能，会自己优化性能

- 坑：位运算符 优先级 比 算术运算符 低

bool(x): returns the Boolean value of a specified object
Any nonzero number is interpreted as “True”
Any nonempty string is “True”
Return False for [], (), {}, False, 0, None

Exercise

- Given a positive integer n, count how many zeros at its end
- The Hamming distance between two integers is the number of positions at which the corresponding bits are different (即对应的二进制表示，有多少位不同)

  Given two integers, calculate the Hamming distance.

  Input: x=1, y = 4

  Output: 2

Function

Summary: Function call in python

Copied from lecture 3

```
6   def f(x, a, b, c):
7       return a * x**2 + b * x + c
8
9   def my_print(msg):
10      print("$ ", end='')
11      print(msg, end='')
12      print(" $")
13
14  z = f(1, 1, 1, 1)
15  print(z)
16
17  my_print('hello world')
```

- 系统定义的函数(譬如print(), int())和用户定义的函数(f(x,a,b,c))，定义的时候，函数本身并不会被执行，只有调用的时候才会执行
- 函数被调用的时候（譬如我们调用print()(或者f(1,1,1,1))），程序会跳转到被调用函数的定义，从函数头开始执行
- 如果函数有参数，那么我们调用的时候参数会被传到函数头的参数。也就是函数头的参数会被初始化赋值
- 函数体的语句会一条一条的顺序执行，直到结束。函数运行结束后，系统会从函数体跳转回到程序原来调用函数的地方。对于需要返回值的函数, 系统通过return 把返回值返回给调用者；对于没有返回值的函数，系统会自动返回
- 对于有return的函数，函数调用可以作为一个值来使用。没有return的，系统会默认返回None，也就是空
  - return会把程序运行的地点从函数体转移回函数调用的地方。无论return后面有没有语句，都不会被执行了。
  - return命令的效果就是从函数的运行返回到函数调用的地方。如果需要返回一个计算值，那么用return xxxx; 如果不需要返回计算值，可以直接一个return
- 一般情况下，函数的定义中使用的变量，不会对外面定义的变量有干涉：因为他们属于不同的势力范围

首先，我们定义了两个函数f和my_print，函数定义本身并不会被执行。我们调用f(1,1,1,1)的时候，系统会跳转到f的定义的部分(也就是def f)，开始运行：首先参数x,a,b,c会被赋值为1,1,1,1；然后函数体中的语句会被执行，直到计算出y。通过return y语句，系统跳转回原来的语句z=f(1,1,1,1), 并且将return 回来的y赋给了z。下面是一个函数调用更复杂的例子：函数调用了四次，return了四个值

```
w = f(1,1,1,1) + f(1,2,3,4) + f(-2,-1,0,1)*f(5,6,7,8)
print(w)
```

Locality (局部性)

- Variables and parameters of functions are local （互不干涉）
  - When you create a local variable inside a function, it only exists inside the function, and you cannot use it outside:
  - Parameters are also local

```
def concate(str1, str2):
    str3 = str1 + str2
    return str3

print(str3)
```

```
Traceback (most recent call last):
  File "C:\Users\fcheng\OneDrive\CS124计算导论\2018\lecture notes\2.py", line 5, in <module>
    print(str3)
NameError: name 'str3' is not defined
```

str3 can be used only inside the function

```
str1 = "hello"
str2 = "world"
str3 = "2019"

def concate(str1, str2):
    str3 = str1 + str2
    return str3

print(concate(str1, str2))
print(str3)
```

```
helloworld
2019
```

global: break locality

global (全局): 如果要在函数内修改函数外部的某个变量，可以在前面加global关键字

- 可以在函数内部，直接使用外部的变量，如果不修改这个变量

```
799  s = 0
800
801  def add(x):
802  #     s = 100
803      s += x
804
805  for x in range(100):
806      add(x)
807
808  print(s)
```

```
799  s = 0
800
801  def add(x):
802      s = 100
803      s += x
804
805  for x in range(100):
806      add(x)
807
808  print(s)
```

```
0
```

```
799  s = 0
800
801  def add(x):
802      global s
803      s += x
804
805  for x in range(100):
806      add(x)
807
808  print(s)
```

```
4950
```

Default parameter (默认参数)

- If we call the function without parameter, it uses the default value:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

```
def f(x=1, a=1, b=0, c=0):
    return a*x**2 + b*x + c

print(f())
print(f(2))
print(f(2,-1))
print(f(2,-1,1))
print(f(2,-1,1,1))
```

```
1
4
-4
-2
-1
```

- Passing via keyword. In general, we need to align parameters by order. We can avoid it by passing via keyword

```
print(f(x=3), f(a=-1), f(a=1, x=1), f(c=-1, b=1, x= -7))
```

```
9 -1 1 41
```

- In this example, we use assign the parameters the values. It can help us get rid of the mistakes of misalignment

推荐使用：不犯错误的唯一办法是把错误本身消灭掉

*args, **kwargs

*args: Arbitrary positional arguments. **kwargs: Arbitrary keyword arguments

- For arbitrary positional argument, an asterisk (*) is placed before a parameter in function definition which can hold non-keyword variable-length arguments. (Tuple)
- For arbitrary positional argument, a double asterisk (**) is placed before a parameter in a function which can hold keyword variable-length arguments. (Dict)

```
def f(*args):
    for x in args:
        print(x)


f(1, 2, 3)
f(3, 4)
f(7)
```

```
1
2
3
3
4
7
```

```
def m(*args):
    z = 1
    for x in args:
        z *= x

    print(z)


m(1, 2, 3)
m(-1, -2)
m(1, 1j, 2j)
```

```
6
2
(-2+0j)
```

```
def test_kwargs(**kwargs):
    for x in kwargs:
        print(kwargs[x], end=' ')

    print()


test_kwargs(x=1)
test_kwargs(x=1, y=2)
test_kwargs(x=1, y=2, z=3)
test_kwargs(x=1, y=2, z=3, u=4)
```

```
1
1 2
1 2 3
1 2 3 4
```

Positional-only parameters (3.8)

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
      -----------    ----------     ----------
        |             |                  |
        |        Positional or keyword   |
        |                                - Keyword only
         -- Positional only
```

- A function definition may look like:
- where / and * are optional. If used, these symbols indicate the kind of parameter by how the arguments may be passed to the function: positional-only, positional-or-keyword, and keyword-only. Keyword parameters are also referred to as named parameters

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)

f(10, 20, 30, d=40, e=50, f=60)
# f(10, b=20, c=30, d=40, e=50, f=60)   # b cannot be a keyword argument
# f(10, 20, 30, 40, 50, f=60)           # e must be a keyword argument
```

```
print(divmod(100, 33))
print(divmod(b=100, a=33)) #TypeError: divmod() takes no keyword arguments

def divmod(a, b, /):
    "Emulate the built in divmod() function"
    return (a // b, a % b)
```

https://docs.python.org/3/contents.html

5 arguments

1. default arguments
2. keyword arguments
3. positional arguments
4. arbitrary positional arguments
5. arbitrary keyword arguments

- important points
  - Default should follow non-default
  - Keyword should follow positional

Find the answer yourself

```
  File "c:\Users\popeC\OneDrive\CS124计算导论\2023 秋季\course_code.py", line 37
    def area(a=1, b=1, c):
                       ^
SyntaxError: non-default argument follows default argument
```

```
  File "c:\Users\popeC\OneDrive\CS124计算导论\2023 秋季\course_code.py", line 45
    print(area(a=1, 2, 2))
                  ^
SyntaxError: positional argument follows keyword argument
```

```python
def area(a=1, b=1, c=1):
    return ((q:=(a+b+c)/2) * (q-a) * (q-b) * (q-c))**0.5

print(area())
```

```python
def area(a, b=1, c=1):
    return ((q:=(a+b+c)/2) * (q-a) * (q-b) * (q-c))**0.5

print(area(1))
```

```python
def area(a=1, b=1, c):
    return ((q:=(a+b+c)/2) * (q-a) * (q-b) * (q-c))**0.5

print(area(c=1))
```

```python
def area(a=1, b=1, c=1):
    return ((q:=(a+b+c)/2) * (q-a) * (q-b) * (q-c))**0.5

print(area(a=1, 2, 2))
```

Indentation (缩进)

- 缩进用来表示命令属于哪个语句块(block)，相同的缩进就表明在一个语句块
- Python中，if, while, for, def, class 等语句会和下面的语句语句块构成一个整体。缩进会表明这个这条语句是不是和上面的语句头构成一个整体

```
times = 0
for i in range(3):
    for j in range(3):
        times += 1
        print("Times = {}".format(times)) # 和for j一个整体

print(times)
```

```
times = 0
for i in range(3):
    for j in range(3):
        times += 1
print("Times = {}".format(times))# 和for i 同级别

print(times)
```

```
times = 0
for i in range(3):
    for j in range(3):
        times += 1
    print("Times = {}".format(times))# 和for i一个整体

print(times)
```

```
Times = 1
Times = 2
Times = 3
Times = 4
Times = 5
Times = 6
Times = 7
Times = 8
Times = 9
9
Times = 3
Times = 6
Times = 9
9
Times = 9
9
```

编程规范：for + for不超过两层。
特别是有return, break的情况下。
超过两层，第三层的逻辑用函数实现

Indentation: common errors

- Don’t use “Tab” and “whitespace” simultaneously
- Abide by the rule of indentation for each statement strictly
- Don’t add any extra whitespace before indentation

```
def f(x, a, b, c):
    y = a*x**2 + b*x +c

    if y ==1:
        print("test")

    return y
```

```
  File "C:\Users\fcheng\Desktop\test3.py", line 5
    print("test")
            ^
TabError: inconsistent use of tabs and spaces in indentation
```

```
def f(x, a, b, c):
    y = a*x**2 + b*x +c
return y
```

```
def f(x, a, b, c):
    y = a*x**2 + b*x +c
     z = a*x**2 + b*x +c

    return y
```

“Whitespace” is preferred than “Tab”

Functions with the same name

- Python中，函数必须在使用前定义，后面的函数可以调用前面的函数
- Python中，两个同名函数， 无论参数是否相同，后面的函数会覆盖前面的函数。即，python中没有重载，只有重写

```python
def f1(n):
    return n * n

def f2(n):
    return f1(n) + 2*n + 1

def f3(n):
    return f2(n) + f1(n) + 1

print(f3(100), f3(10), f3(1))
```

```
20202 222 6
```

```python
def func1(name, age):
    print(f"name = {name}, age = {age}")


func1("Hello", 28)


def func1(name, age, country):
    print("name = {name}, age = {age}, country = {country}")


# func1("Hello", 28) # error
func1("Hello", 28, "CN")
```

```
name = Hello, age = 28
name = Hello, age = 28, country = CN
```

Type Hint：类型提示 (Python 3.5+)

- In Python, we don’t need to specify the types of arguments and return values as C++ and Java do
- However, it is convenient to specify the types if possible

```python
def add(x: float, y: float) -> float:
    return x + y

print(add(1, 3))
print(add("1", "3"))
```

```python
def add(x: float, y: float) -> float:
    return x + y


print(add(1, 3))
print(add("1", "3"))
```

- You need to install an extension
- pip install mypy
- >>mypy .\course_code.py
- mypy extension in vscode

```python
primes: list[int] = []

captain: str  # Note: no initial value!

class Starship:
    stats: dict[str, int] = {}
```

```
PS C:\Users\popeC\OneDrive\CS124计算导论\2022 秋季\lecture notes> mypy .\course_code.py
course_code.py:12: error: Argument 1 to "add" has incompatible type "str"; expected "float"  [arg-type]
course_code.py:12: error: Argument 2 to "add" has incompatible type "str"; expected "float"  [arg-type]
Found 2 errors in 1 file (checked 1 source file)
```

https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html

动态类型是一个把双刃剑，对自由的滥用

Programming with style

- Readability is very important to programmers, since in practice programs are read and modified far more often than they are written.
- A recommended programming style for Python is shown as follows:
  - use 4 spaces for indentation
  - imports should go at the top of the file
  - separate function definitions with two blank lines
  - keep function definitions together
  - keep top level statements, including function calls, together at the bottom of the program
- More programming styles for Python can be obtained from the website
  http://www.python.org/dev/peps/pep-0008/ (Style Guide for Python Code)
  推荐 安装VS Code Extension，右键选择 format document

编程规范：不包含注释，单个函数的代码长度不要超过40行。超过这个长度，要么切割成更小的几个函数，要么你的思路有问题，要重新写