计算思维通识教育

Computational Thinking

第2章 计算设备处理信息-使用编程语言

主讲人：曹轶臻

联系方式：caoyizhen@cuc.edu.cn

计算思维用 分解和抽象 迎战 庞大与复杂!

CONTENTS

计算机与网络空间安全学院
School of Computer and Cyber Sciences

计算思维通识教育
Computational Thinking

03

编程语言的结构与交互

Python

计算机与网络空间安全学院
School of Computer and Cyber Sciences

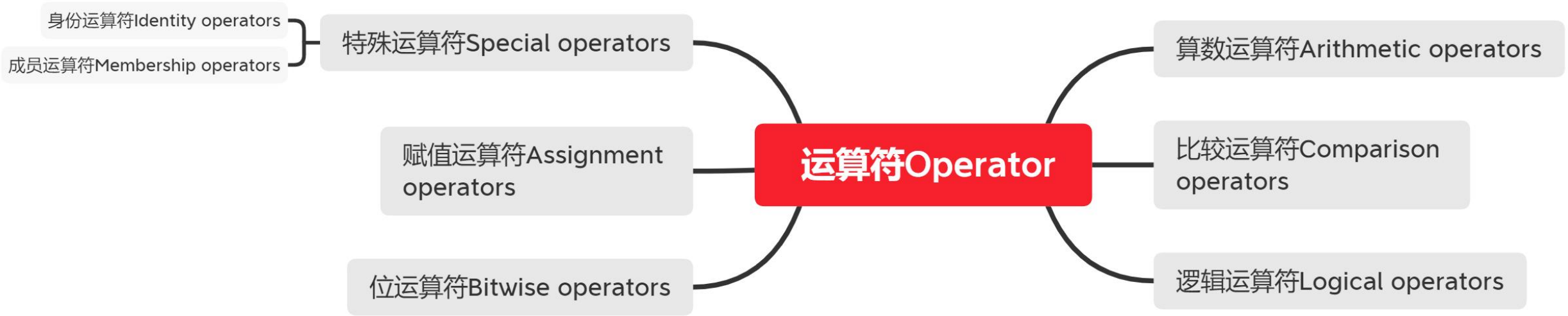
3-1 运算符

运算符（Operator）是 编程语言中执行算术或逻辑运算的特殊符号。

运算符操作的值称为操作数。

```
>>> 2+3
5
```

+ 是执行加法的运算符。
2 和 3 是操作数，
5 是操作的输出。



3-1 运算符

1.算数运算符

算术运算符用于执行数学运算，如加法、减法、乘法等。

算术运算符	含义
+	加法或一元加
-	减法或二元减
*	乘法
/	除法 (结果是float类型)
%	取余
//	地板除（Floor division）向下取整，返回商的整数部分
**	幂 - 返回x的y次幂

```
x = 15
y = 4
# Output: x + y = 19
print('x + y =',x+y)
# Output: x - y = 11
print('x - y =',x-y)
# Output: x * y = 60
print('x * y =',x*y)
# Output: x / y = 3.75
print('x / y =',x/y)
# Output: x // y = 3
print('x // y =',x//y)
# Output: x ** y = 50625
print('x ** y =',x**y)
```

3-1 运算符

2.比较运算符

用于比较值，根据条件返回 True 或 False。

比较运算符	含义
>	大于 - 如果左操作数大于右操作数，则为真
<	小于 - 如果左操作数小于右操作数，则为真
==	等于 - 如果两个操作数相等则为真
!=	不等于 - 如果操作数不相等则为真
>=	大于等于 - 如果左操作数大于或等于右操作数，则为真
<=	小于等于 - 如果左操作数小于或等于右操作数，则为真

```
x = 10
y = 12
# Output: x > y is False
print('x > y is',x>y)
# Output: x < y is True
print('x < y is',x<y)
# Output: x == y is False
print('x == y is',x==y)
# Output: x != y is True
print('x != y is',x!=y)
# Output: x >= y is False
print('x >= y is',x>=y)
# Output: x <= y is True
print('x <= y is',x<=y)
```

3-1 运算符

3.逻辑运算符

逻辑运算符	含义
and	布尔“与”，如果两个操作数都为真，则为真
or	布尔“或”，如果任一操作数为真，则为真
not	布尔“非”，如果操作数为真，返回False，反之返回True

Truth table for logical operators				
A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

```
x = True
y = False

# Output:x and y is False
print('x and y is', x and y)

# Output:x or y is True
print('x or y is', x or y)

# Output:not x is False
print('not x is', not x)
```

3-1 运算符

4.位运算符

位运算符把数字看作二进制，一位一位地进行计算。

位运算符	含义
&	按位与运算符
|	按位或运算符
~	按位取反运算符
^	按位异或运算符
<<	左移动运算符：运算数的各二进位全部左移若干位，由 << 右边的数字指定了移动的位数，高位丢弃，低位补0。
>>	右移动运算符：把">>"左边的运算数的各二进位全部右移若干位，>> 右边的数字指定了移动的位数.

```
x = 10 # 00001010
y = 4  # 00000100

print('x & y is', x & y, bin(x & y))
print('x | y is', x | y, bin(x | y))
print('x ^ Y is', x ^ y, bin(x ^ y))
print('~x is', ~x, bin(~x))
print('x >> 2 is', x >> 2, bin(x >> 2))
print('x << 2 is', x << 2, bin(x << 2))
```

```
x & y is 0 0b0
x | y is 14 0b1110
x ^ Y is 14 0b1110
~x is -11 -0b1011
x >> 2 is 2 0b10
x << 2 is 40 0b101000
```

3-1 运算符

5.赋值运算符

Python 中使用赋值运算符为变量赋值。

Python 中有各种复合运算符，

例如a += 5，等价于 a = a + 5。

赋值运算符	示例	含义
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
其它复合运算符		

```
a, b, c = 21, 10, 0

c = a + b
print('c =',c)
c += a
print('c =',c)
c *= a
print('c =',c)
c /= a
print('c =',c)

c = 2
c %= a
print('c =',c)
c **= a
print('c =',c)
c //= a
print('c =',c)
```

```
c = 31
c = 52
c = 1092
c = 52.0
c = 2
c = 2097152
c = 99864
```

3-1 运算符

6.特殊运算符

成员运算符，用于测试是否能在序列（字符串、列表、元组、集合和字典）中找到值或变量。

成员运算符	含义
in	如果在序列中找到值返回 True，否则返回 False。
not in	如果在序列中没有找到值返回 True，否则返回 False。

```
student_name = 'Jules'
marks = {'James': 90, 'Ford': 86, 'Jules': 55, 'Arthur': 77}

for student in marks:
    if student == student_name:
        print(marks[student])
        break
```

```
student_name = 'Jules'
marks = {'James': 90, 'Ford': 86, 'Jules': 55, 'Arthur': 77}

if student_name in marks:
    print(marks[student_name])
```

用成员运算符in改写

```
x = 'Hello world'
y = {1:'a',2:'b'}
z = [1,2,3,4,5]

# Output: True
print('hello' not in x)

# Output: True
print(1 in y)

# Output: False
print(3 not in z)
```

3-1 运算符

6.特殊运算符

身份运算符,用于检查两个值（或变量）是否位于内存的同一位置。

身份运算符	含义
is	如果操作数相同则为True（引用自同一个对象）。
is not	如果操作数不相同则为True（不引用自同一个对象）。

```
student_1 = 'James'
student_2 = 'James'
student_3 = student_1

print(student_1 == student_2)
print(student_2 is student_1)
print(student_3 is student_1)
print(id(student_1),id(student_2),id(student_3))
```

```
class_1 = ['James','Thomas','Vespa']
class_2 = ['James','Thomas','Vespa']
class_3 = class_1

print(class_1 == class_2)
print(class_2 is class_1)
print(class_3 is class_1)
print(id(class_1),id(class_2),id(class_3))
```

两个相等的变量并不意味着它们是相同的，id函数可以返回变量在内存中的地址

Python 程序的结构

一个简单的程序

```
answer = 0
increament = 1

number = int(input())
while abs(answer**2 - number) > 0.001:
    if answer**2 < number:
        answer += increament
    elif answer**2 > number:
        answer -= increament
        increament /= 10.0

print(number,'的平方根是',answer)
```

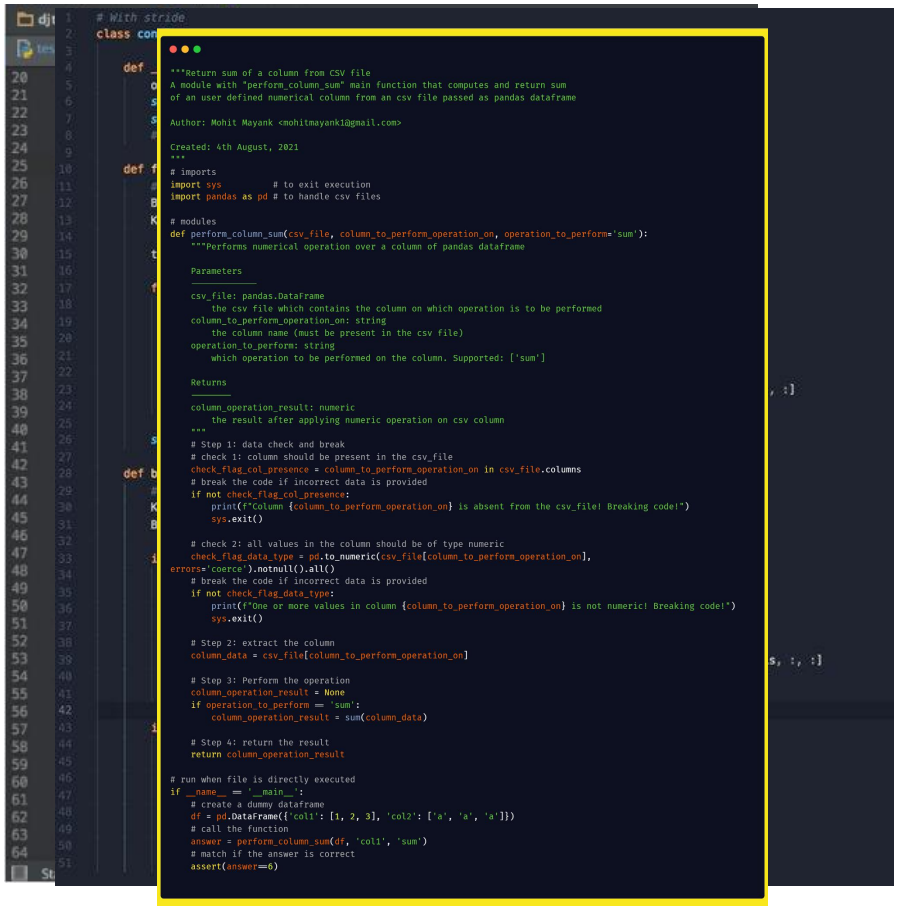
```
import math

number = int(input())
print(number,'的平方根是',math.sqrt(number))
```

现实世界的程序很复杂（成千上万行），每一行代码都要从头编写吗？

如何高效地实现和管理？

计算思维中的“问题分解”：
-- 利用模块化编程（一种将大型编码任务分解为更小且更易于管理的子任务的方法）

一个复杂的程序



这就是 Python 有这么多模块（module）、包（package）、库（library）和框架（framework）的原因。

Python

程序的结构

Python在不同层次的结构划分，将复杂度分解到不同的部分（库/包、模块、函数）去处理。框架通常比库更复杂，除了包含执行特定操作的包，还包含应用程序的基本流程和架构。

扩展阅读：Difference Between Python Modules, Packages, Libraries, and Frameworks

```
cody@cody-Serval-WS ~/PycharmProjects/VirtualPLC
File  Edit  View  Search  Terminal  Tabs  Help
cody@cody-Serval-WS ~/PycharmProjects/...   cody@cody-Serval-WS ~/PycharmProjects/...
├── __init__.py
├── Models
│   ├── FuelFarm
│   │   ├── components.py
│   │   ├── functionality.py
│   │   ├── hmi
│   │   │   ├── fuel_schematic.png
│   │   │   ├── hmi.kv
│   │   │   ├── hmilayout.py
│   │   │   └── __init__.py
│   │   └── __init__.py
│   └── __init__.py
├── PipingSystems
│   ├── __init__.py
│   ├── pump
│   │   ├── __init__.py
│   │   └── pump.py
│   ├── storage_tank
│   │   ├── __init__.py
│   │   └── tank.py
│   ├── user_creation_script.py
│   └── valve
│       ├── __init__.py
│       └── valve.py
├── tests
│   ├── __init__.py
│   ├── models
│   │   ├── fuel_farm
│   │   │   ├── __init__.py
│   │   │   └── test_fuel_components.py
│   │   ├── hmi
│   │   │   └── __init__.py
│   │   └── piping
│   │       ├── initial_conditions.py
│   │       ├── __init__.py
│   │       ├── test_centrifPump.py
│   │       ├── test_change_values.py
│   │       ├── test_gate.py
│   │       ├── test_generic_pump.py
│   │       ├── test_generic_valve.py
│   │       ├── test_globe.py
│   │       ├── test_parameters.py
│   │       ├── test_positiveDisplacement.py
│   │       ├── test_relief.py
│   │       └── test_tank.py
├── Utilities
│   ├── __init__.py
│   └── utility_formulas.py
└── venv
    ├── bin
    │   ├── python
    │   └── python3
    └── pyvenv.cfg

15 directories, 38 files
cody@cody-Serval-WS ~/PycharmProjects/VirtualPLC $
```

3-2 函数与调用

函数 Function

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。

函数能提高应用的模块性，和代码的重复利用率。

Python提供了许多内建函数，比如print()、input()、int()等等，可以随时使用。

Built-in Functions
https://docs.python.org/3/library/functions.html

Built-in Functions

A	E	L	R
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	F	M	
ascii()	filter()	map()	S
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	G	N	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	H	object()	
callable()	hasattr()	oct()	T
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		P	V
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	Z
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		_
	iter()		__import__()

3-2 函数与调用

函数 Function

也可以自己创建函数，这叫做用户自定义函数（User-defined function）

唯一标识函数的函数名

将值传递给函数的参数（它们是可选的）

关键词def标记函数头的开始

定义函数的语法

```
def function_name(parameters):
    statement(s)
```

冒号 (:) 标记函数头的结束

构成函数体的一个或多个有效的 Python 语句。语句必须具有相同的缩进级别（通常为 4 个空格）

```
def greet(name):
    print("Hello, " + name + ". Good morning!")
```

```
>>> greet('Paul')
Hello, Paul. Good morning!
```

3-2 函数与调用

函数 Function

return 语句结束函数，选择性地返回一个值给调用方，并返回到调用它的地方。

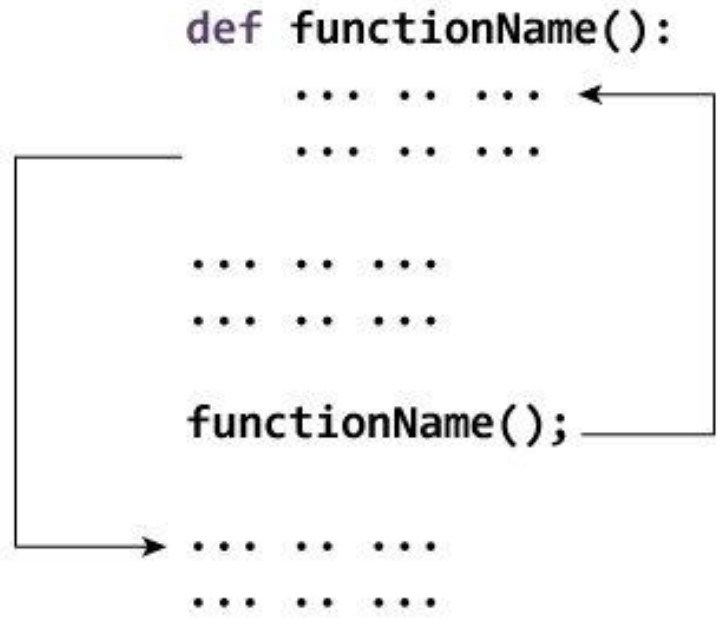
不带return语句的函数相当于返回 None。

```
def greet(name):
    return "Hello, " + name + ". Good morning!"
```

```
>>> print(greet("May"))
Hello, May. Good morning!
```

```
def absolute_value(num):
    if num >= 0:
        return num
    else:
        return -num

print(absolute_value(2))
print(absolute_value(-4))
```

一旦我们定义了一个函数，我们就可以从另一个语句、函数、程序甚至 Python 提示符中调用(call)它。要调用函数，只需键入带有适当参数的函数名称。



函数在 Python 中是如何工作的?

3-2 函数与调用

函数 Function

变量的作用范围是识别变量的程序部分，在函数内部定义的参数和变量在函数外部不可见。

局部变量

```
def my_func_1(a,b):
    a += 10
    b += 10
    print("Values inside function:",a,b)

def my_func_2(a,b):
    product = a * b
    print("product:", product)

a , b = 100,200
my_func_1(a,b)
print("Values outside function:",a,b)

my_func_2(a,b)
print("prodcut:",product)
```

全局变量

```
Values inside function: 110 210
Values outside function: 100 200
product: 20000
Traceback (most recent call last):
  File "C:\Users\cyz\PycharmProjects\pythonProject\test.py", line 13, in
<module>
    print("prodcut:",product)
```

一旦我们从函数返回，它们就会被销毁。
因此，函数不会记住之前调用的变量值。

3-3 模块、包与导入

模块与导入

为了编写可维护可重用的代码，通常把代码按功能分类， 分别放在不同的文件里，这样每个文件中的代码就相对较少，且功能统一。 在Python中，一个 .py 脚本源码文件就称之为一个模块 （module）。

```
# module0.py
module_name = 'module0'


def module_info():
    print("func in module0")
```

```
# module1.py
module_name = 'module1'


def module_info():
    print("func in module1")
```

用import导入模块

```
# test_module.py
import module0, module1 # 导入模块

print(module0.module_name)
print(module1.module_name)
module0.module_info()
module1.module_info()
```

- 使用模块还可以避免函数名和变量名冲突。
- 同名函数和变量可以同时存在不同的模块中，因此在编写模块时， 不必考虑名字会与其他模块冲突
- 这在多人协同编程时至关重要。

3-3 模块、包与导入

模块与导入

import 将整个模块的全局符号表导入到当前模块的符号表中，可以使用模块中所有全局变量，类和函数。

重命名模块

```
import module0 as m0
import module1 as m1

print(m0.module_name)
print(m1.module_name)
m0.module_info()
m1.module_info()
```

import 导入的模块，对其中的变量和函数访问时要加上模块名

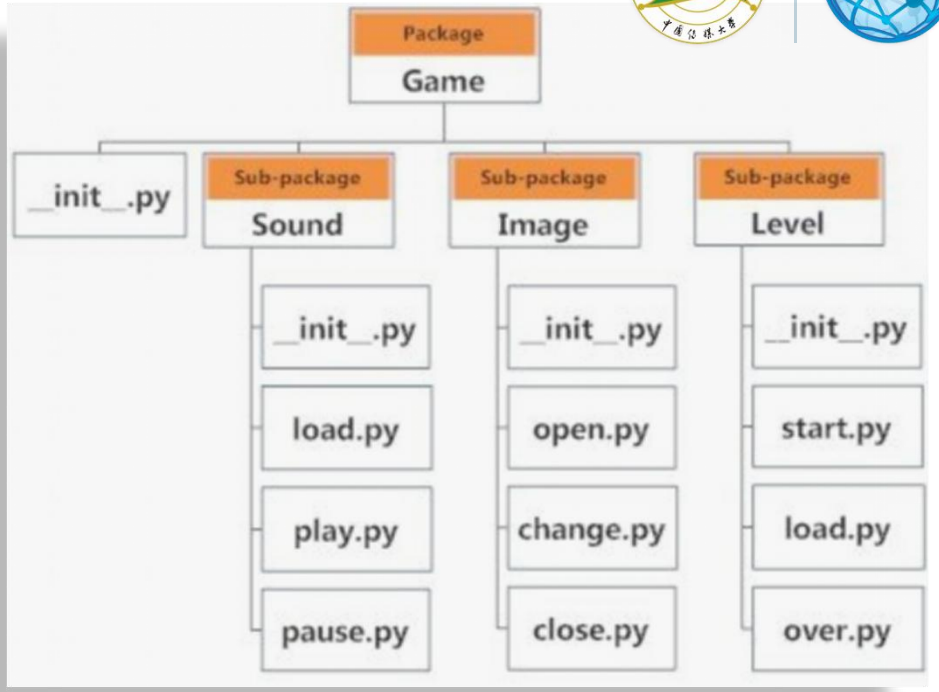
模块部分导入

```
from module0 import module_info
from module1 import module_info as mi

#output:func in module0
module_info()
#output:func in module1
mi()
#NameError: name 'module_name' is not defined
print(module_name)
```

3-3 模块、包与导入

包与导入

- 应用程序越来越大，模块越来越多
- 我们将相似的模块放在一个包(Package)中，将不同的模块放在不同的包中
- 使项目（程序）易于管理且概念清晰



我们使用组织良好的目录层次包结构以便于访问， Python 包可以包含子包和模块。

一个包的目录必须包含 __init__.py 文件。

```
from Game.Level import start

start.select_difficulty(2)
```

```
from Game.Level.start import select_difficulty

select_difficulty(2)
```

Also

导入项目中的启动模块，可以这样做：

```
import Game.Level.start

Game.Level.start.select_difficulty(2)
```

使用点 (.) 运算符从包中导入模块

扩展阅读：Most Popular Python Packages

3-4 输入输出

输出 Output

内置函数input() 和 print() 等广泛用于标准输入和输出操作。

print() 函数的实际语法是

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='*')
print(1, 2, 3, 4, sep='#', end='&')
```

```
1 2 3 4
1*2*3*4
1#2#3#4&
```

在输出时，常常使用 str.format() 或f-string方法来完成输出的格式化（令其更好看）

```
>>> print('I love {0} and {1}'.format('bread','butter'))
I love bread and butter
>>> print('I love {1} and {0}'.format('bread','butter'))
I love butter and bread
```

```
>>> x,y = 5,10
>>> print(f'The value of x is {x} and y is {y}')
The value of x is 5 and y is 10
```

3-4 输入输出

输入 Input

如果我们希望从用户那里获取输入，用input()函数来实现。

input() 函数的实际语法是

```
input([prompt])
```

其中prompt是我们希望在屏幕上显示的字符串（它是可选的）

```
>>> num = input('Enter a number: ‘)
Enter a number: 10
>>> num
’10’
>>> num + 1
TypeError: can only concatenate str (not "int") to str
```

```
names = []
for i in range(5):
    n = input(f'请输入第{i+1}个同学的姓名：')
    names.append(n)

print(names)
```

本次课程结束，请继续加油！

Computational Thinking

计算机与网络空间安全学院
School of Computer and Cyber Sciences