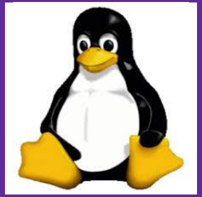
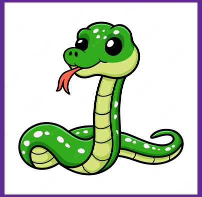
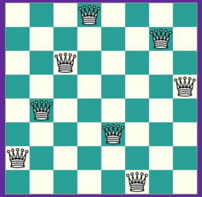
Introduction to Computation

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

3

Outline

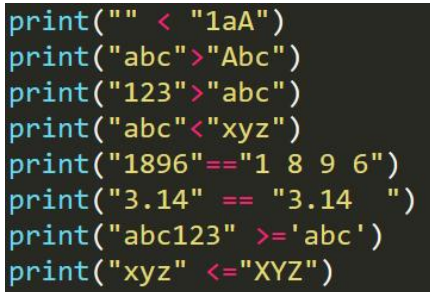
- String Processing
- Usage of print()
- Function

String Processing

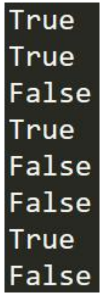
String: >, =, <

- In practice, letters are ordered: **‘a’<‘b’<‘c’<…<‘z’**
- In ASCII, characters are ordered by their ASCII:

  ‘\n’<‘0’<‘9’<‘A’<‘Z’<‘a’<‘z’
- In practice, letters are ordered: **‘a’<‘b’<‘c’<…<‘z’**
- 中文姓名排序：“张一一” > “张一二”，“张一” > “张二一一一”
- **Alphabetical order** (字典序)： For two strings, their first letters are compared. If they differ, then the string whose first letter comes earlier in the alphabet comes before the other string
  - If the first letters are the same, then the second letters are compared, and so on
  - If a position is reached where one string has no more letters to compare while the other does, then the first (shorter) string is deemed to come first in alphabetical order
- In python, strings are compared by alphabetical order
  - >=, <=, ==, >, <, !=



```
print("" < "1aA")
print("abc">"Abc")
print("123">"abc")
print("abc"<"xyz")
print("1896"=="1 8 9 6")
print("3.14" == "3.14  ")
print("abc123" >='abc')
print("xyz" <="XYZ")
```



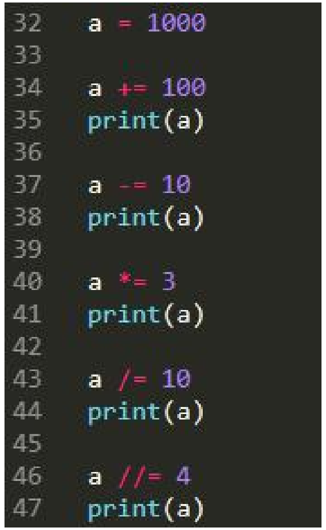
```
True
True
False
True
False
False
True
False
```

How to get ASCII of ‘x’

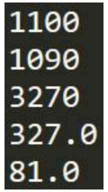
https://en.wikipedia.org/wiki/Alphabetical_order

+=, /=, *=

- In python, we could rewrite an expression like
  a = a + b as a += b
- a = a + b, a += b
- a = a – b, a -= b
- a = a * b, a *= b
- a = a / b, a /= b
- a %= b, a //= b



```
a = 1000

a += 100
print(a)

a -= 10
print(a)

a *= 3
print(a)

a /= 10
print(a)

a //= 4
print(a)
```

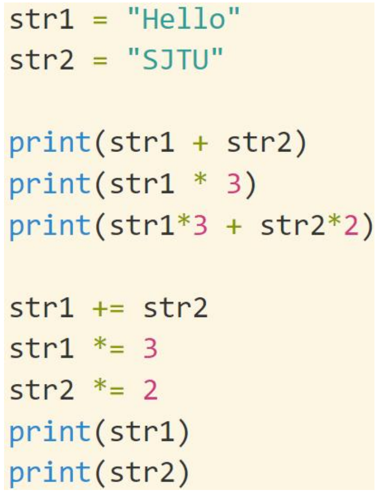


```
1100
1090
3270
327.0
81.0
```

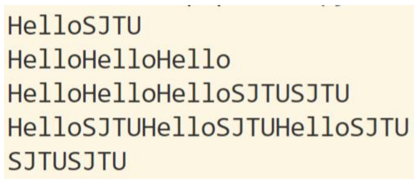
能够用 自操作符 的地方尽量用
x = x+1
x += 1 (推荐)

str: +, *, +=, *=

- str1 + str2： return the concatenation of str1 and str2 (连接)
- str1* n: return a new string that repeats str1 for n times



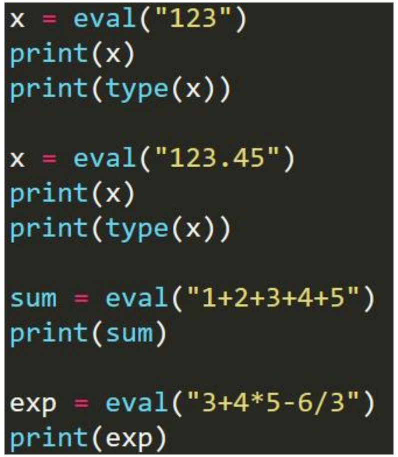
```python
str1 = "Hello"
str2 = "SJTU"

print(str1 + str2)
print(str1 * 3)
print(str1*3 + str2*2)

str1 += str2
str1 *= 3
str2 *= 2
print(str1)
print(str2)
```



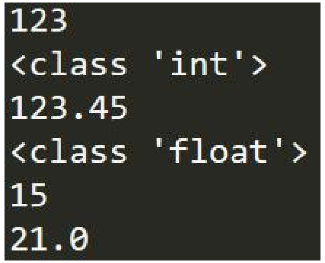
```
HelloSJTU
HelloHelloHello
HelloHelloHelloSJTUSJTU
HelloSJTUHelloSJTUHelloSJTU
SJTUSJTU
```

eval(): evaluate a string

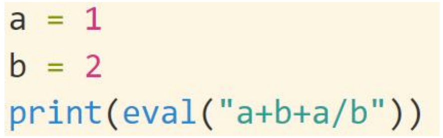
- eval() (**evaluate**) will interprets a string as a Python expression and evaluate its value
  - The grammar is: <variable> = eval(<str>)



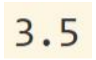
```
x = eval("123")
print(x)
print(type(x))

x = eval("123.45")
print(x)
print(type(x))

sum = eval("1+2+3+4+5")
print(sum)

exp = eval("3+4*5-6/3")
print(exp)
```



```
123
<class 'int'>
123.45
<class 'float'>
15
21.0
```



```
a = 1
b = 2
print(eval("a+b+a/b"))
```

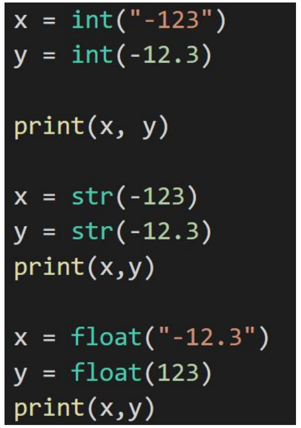


```
3.5
```

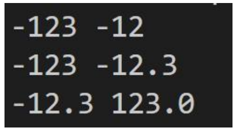
Type conversion

- Four important functions for type issues: **type(), int(), float() and str()**
- int(str): will transform a string/float to integer
  - price = int(“124”)
- float(str): will transform a string/int to float number
  - pi = float(“3.1415926”)
- str(x): will transform an int/float to string
- Float is not accurate
- Don’t abusing these functions
  - int() can only convert an int in string form
  - A float in string form must be converted by float() first and then converted by int()
  - Int(‘3.14’) : error
  - int(float(‘3.14’)): correct, return 3

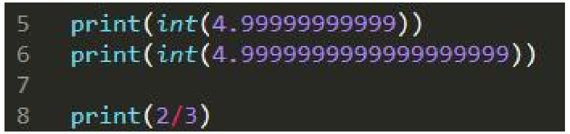
https://docs.python.org/3/tutorial/floatingpoint.html



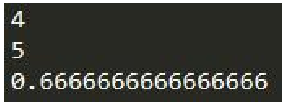
```
x = int("-123")
y = int(-12.3)

print(x, y)

x = str(-123)
y = str(-12.3)
print(x,y)

x = float("-12.3")
y = float(123)
print(x,y)
```



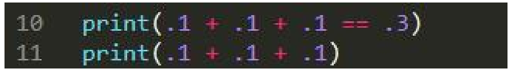
```
-123 -12
-123 -12.3
-12.3 123.0
```



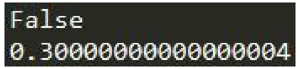
```
print(int(4.99999999999))
print(int(4.9999999999999999999))

print(2/3)
```



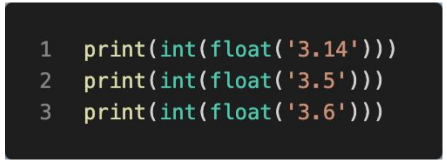
```
4
5
0.6666666666666666
```



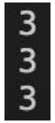
```
print(.1 + .1 + .1 == .3)
print(.1 + .1 + .1)
```



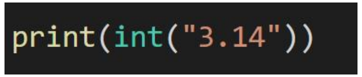
```
False
0.30000000000000004
```



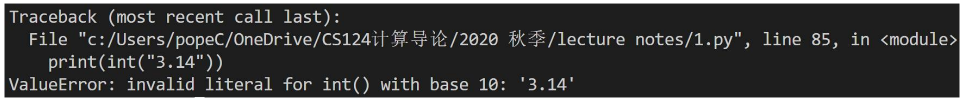
```
print(int(float('3.14')))
print(int(float('3.5')))
print(int(float('3.6')))
```



```
3
3
3
```



```
print(int("3.14"))
```

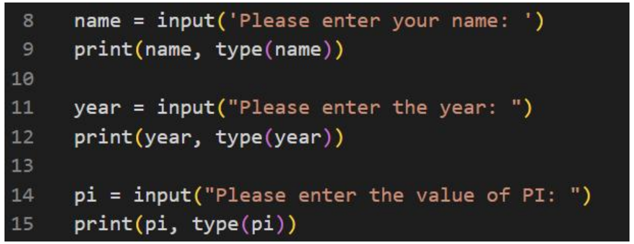


```
Traceback (most recent call last):
  File "c:/Users/popeC/OneDrive/CS124计算导论/2020 秋季/lecture notes/1.py", line 85, in <module>
    print(int("3.14"))
ValueError: invalid literal for int() with base 10: '3.14'
```

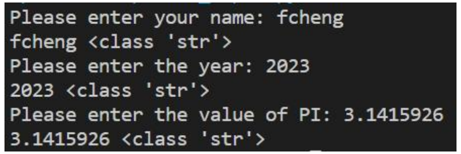
Input from screen: input()

input() will return a string from the characters you typed in screen. “\n” will terminate the input and be ignored.

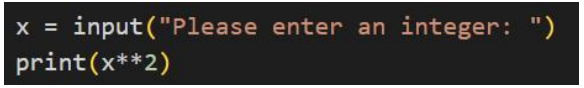
- **No matter what you typed, you will get a string. It is a string！！！(最常见错误).**
- The grammar is: <variable> = input(<prompt>)
  - Here prompt is an expression that serves to prompt the user for input. It is almost always a string literal. Like a hint!



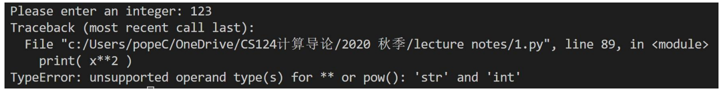
```python
name = input('Please enter your name: ')
print(name, type(name))

year = input("Please enter the year: ")
print(year, type(year))

pi = input("Please enter the value of PI: ")
print(pi, type(pi))
```



```
Please enter your name: fcheng
fcheng <class 'str'>
Please enter the year: 2023
2023 <class 'str'>
Please enter the value of PI: 3.1415926
3.1415926 <class 'str'>
```



```python
x = input("Please enter an integer: ")
print(x**2)
```



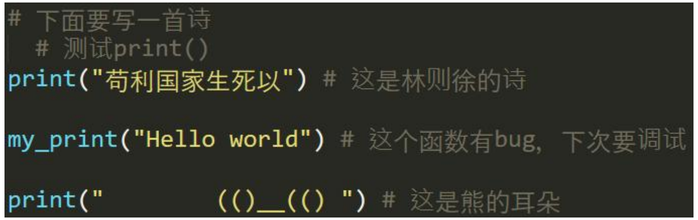
```
Please enter an integer: 123
Traceback (most recent call last):
  File "c:/Users/popeC/OneDrive/CS124计算导论/2020 秋季/lecture notes/1.py", line 89, in <module>
    print( x**2 )
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

TypeError：类型错误

最常见的错误 input()--str

Comment statement 注释语句

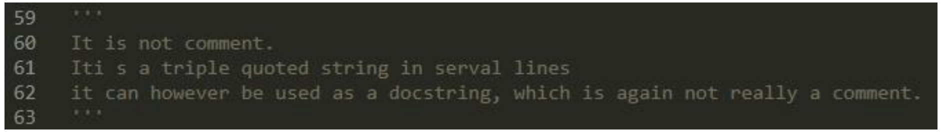
- Sometimes, you need to write some notes for your code. Then you need comment statement
- Comment statement starts with “#” at any position in a line. After it, you are allowed to write down anything
  - Comments will be ignored by the compilers and will not affect the executions of your program
  - Comments will help you and other people to understand your code later
  - It is good to write comments for you program if possible
- In python, there is only one type of comment that starts with hash # and can contain only a single line of text.
- According to PEP 257, Triple quoted strings can however be used as a docstring, which is again not really a comment.



```
# 下面要写一首诗
  # 测试print()
print("苟利国家生死以") # 这是林则徐的诗

my_print("Hello world") # 这个函数有bug, 下次要调试

print("        (()__(() ") # 这是熊的耳朵
```

#后面的语句都会被忽略

#: Sharp
C#



```
'''
It is not comment.
Iti s a triple quoted string in serval lines
it can however be used as a docstring, which is again not really a comment.
'''
```

A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the __doc__ special attribute of that object.

https://peps.python.org/pep-0257/

Usage of print()

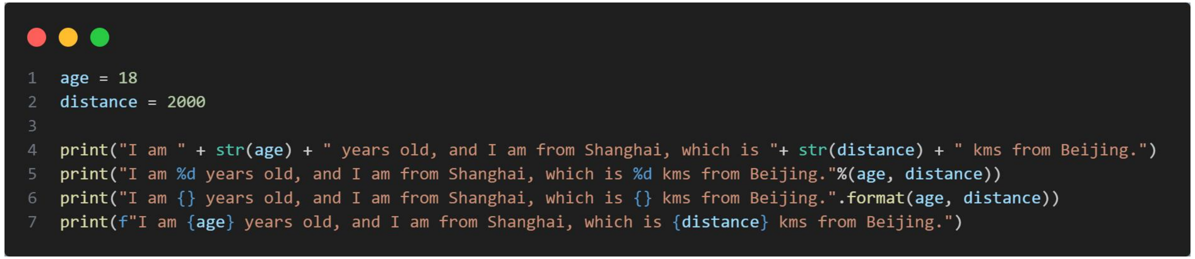
Format print()

- 格式化输出：我们希望在print()输出字符串的时候，加入变量控制的信息

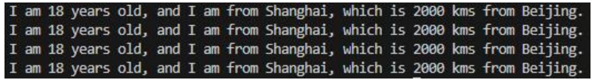
  Question: age = 18   distance = 2000. What if I want to print the following:

  I am 18 years old, and I am from Shanghai, which is 2000 kms from Beijing.

  Here 18 and 2000 is decided by age and distance
- 解决方案：print("I am " + str(age) + " years old, and I am from Shanghai, which is "+ str(distance) + " kms from Beijing.")
  - 手工一一对齐：太繁琐，手工操作，不够直接，容易犯错
- In python, there are three styles to control the format of print()
  - The old style: “%d %d” %(age, distance)（注：来源于C语言）
  - The new style: “{} {}”.format(age, distance)（注：python所特有，更安全）
    - format() 是string类自带的一个函数 ({}中间没有空格)
  - The newer style, f-string: f"{age}...{distance}"



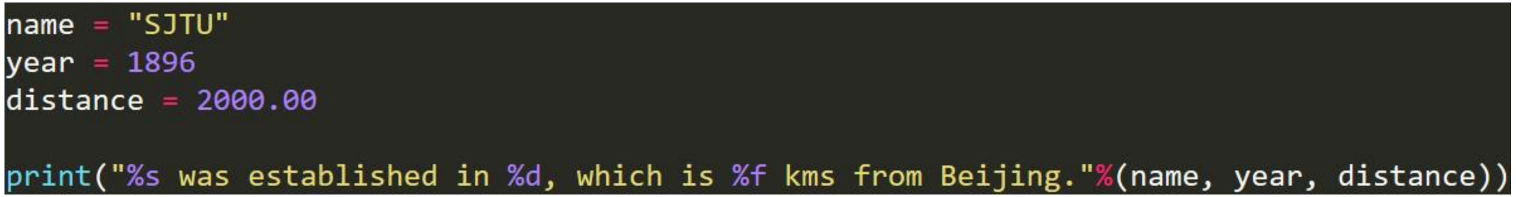
```python
age = 18
distance = 2000

print("I am " + str(age) + " years old, and I am from Shanghai, which is "+ str(distance) + " kms from Beijing.")
print("I am %d years old, and I am from Shanghai, which is %d kms from Beijing."%(age, distance))
print("I am {} years old, and I am from Shanghai, which is {} kms from Beijing.".format(age, distance))
print(f"I am {age} years old, and I am from Shanghai, which is {distance} kms from Beijing.")
```



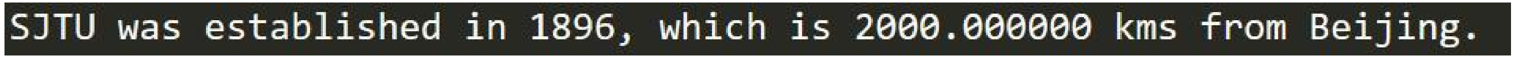
```
I am 18 years old, and I am from Shanghai, which is 2000 kms from Beijing.
I am 18 years old, and I am from Shanghai, which is 2000 kms from Beijing.
I am 18 years old, and I am from Shanghai, which is 2000 kms from Beijing.
I am 18 years old, and I am from Shanghai, which is 2000 kms from Beijing.
```

C-Style print()

- Old style print() is commonly seen in the previous programs. We should understand it.
- In C Language, s: string, d: decimal, f: float
- In Python, % + x: %s (字符串), %d(整数), %f(浮点数)



```python
name = "SJTU"
year = 1896
distance = 2000.00

print("%s was established in %d, which is %f kms from Beijing."%(name, year, distance))
```



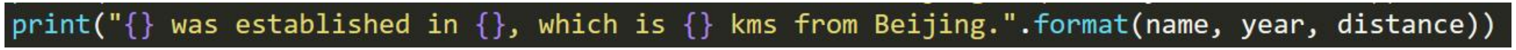
```
SJTU was established in 1896, which is 2000.000000 kms from Beijing.
```

- 问题：name, year, distance 的实际输入类型和%s, %d, %f不一致
  - 历史上计算机软件最多的几种bug
  - 已经不建议使用
  - 老的程序中存在

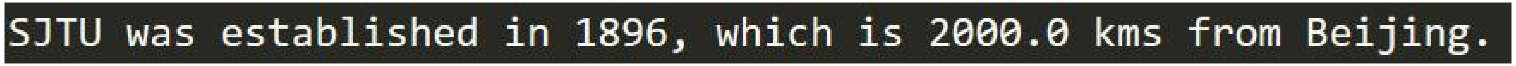
New style with {}

- New Style: {} 占位符， 系统根据输入的数据自动推导其类型

  “{}…{}…{}”.format(par1, par2, par3)



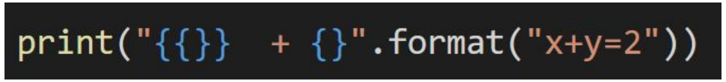
```
print("{} was established in {}, which is {} kms from Beijing.".format(name, year, distance))
```



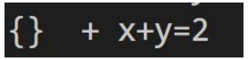
```
SJTU was established in 1896, which is 2000.0 kms from Beijing.
```

- If you need to include a brace character in the literal text, it can be escaped by doubling: {{ and }}.

  print(“{{}}—{ }”.format(“x+y=2”))



```
print("{{}}  + {}".format("x+y=2"))
```



```
{}  + x+y=2
```

- Reference:
  - https://pyformat.info/
  - Old style: https://docs.python.org/2/library/stdtypes.html#string-formatting
  - New style: https://docs.python.org/3/library/string.html#string-formatting

Official documentation is the best assistant of programmers

https://www.w3schools.com/python/ref_string_format.asp

f-String: Formatted string literals

- Python version ≥ 3.6. 推荐使用
- Formatted string literals (also called f-strings for short) let you include the value of Python expressions inside a string by prefixing the string with f or F and writing expressions as {expression}

```
name = "Eric"
age = 74
txt = f"Hello, {name}. You are {age}."
num = f"{2 * 37}"
print(txt)
print(num)
```

```
Hello, Eric. You are 74.
74
```

```
exp1 = f"{70 + 4}"
exp2 = f"{{70 + 4}}"
exp3 = f"{{{70 + 4}}}"
exp4 = f"{{{{70 + 4}}}}"

print(exp1, exp2, exp3, exp4)
```

```
74 {70 + 4} {74} {{70 + 4}}
```

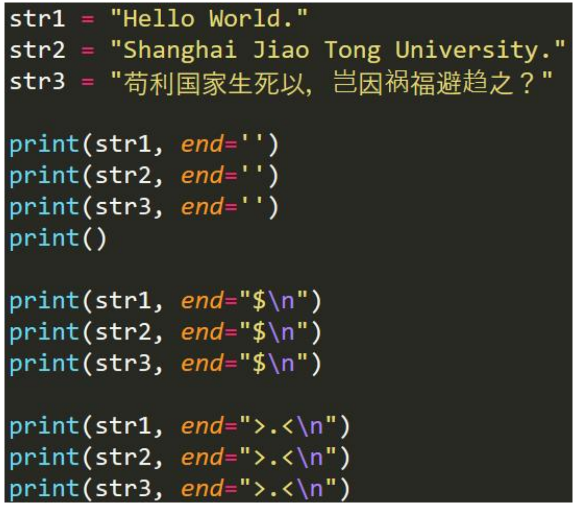
```
import math
print(f'The value of pi is approximately {math.pi:.3f}.')
```

```
The value of pi is approximately 3.142.
```

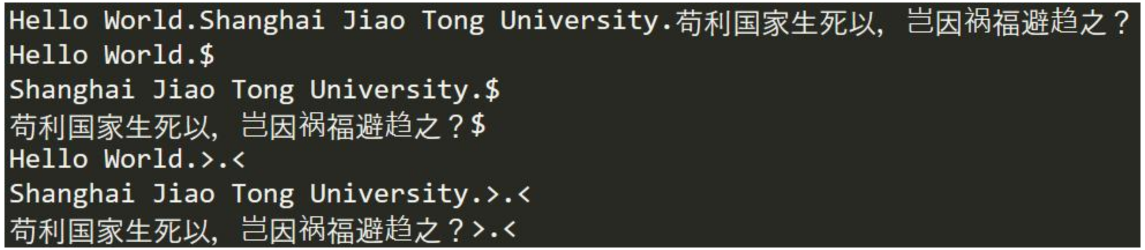
https://docs.python.org/3/tutorial/inputoutput.html

print(): end

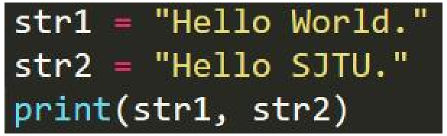
- When you use print(), there will be a newline automatically at the end of the line.
  - The grammar to change it is: print(“something”, end=‘the symbol you like’)
- You can print several data in a line and there will be a single whitespace between them
  - print(str1, str2, st3, …, strn)



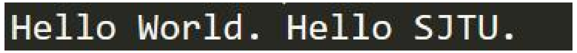
```
str1 = "Hello World."
str2 = "Shanghai Jiao Tong University."
str3 = "苟利国家生死以, 岂因祸福避趋之？"

print(str1, end='')
print(str2, end='')
print(str3, end='')
print()

print(str1, end="$\n")
print(str2, end="$\n")
print(str3, end="$\n")

print(str1, end=">.<\n")
print(str2, end=">.<\n")
print(str3, end=">.<\n")
```



```
Hello World.Shanghai Jiao Tong University.苟利国家生死以, 岂因祸福避趋之？
Hello World.$
Shanghai Jiao Tong University.$
苟利国家生死以, 岂因祸福避趋之？$
Hello World.>.<
Shanghai Jiao Tong University.>.<
苟利国家生死以, 岂因祸福避趋之？>.<
```



```
str1 = "Hello World."
str2 = "Hello SJTU."
print(str1, str2)
```



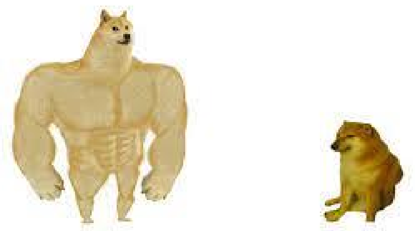
```
Hello World. Hello SJTU.
```

Function

Management of Codes

Function, Class, and Module are the three mechanisms of code management

- 代码规模庞大：单个文件代码长度超过3000行
  - 管理3个人，管理300个人，管理3000个人
- 人员流动性强：程序员中途离职，有新人加入项目
  - Google的某些系统跨度可能有20年，可能刚开始的开发人员都退休了
  - 如何找人代替老员工、新员工如何尽快熟悉业务
- 业务流程复杂，需要多道手续
  - 用户用手机，在饿了么上面下单，商家接单，快递员送货
  - 对象：手机、饿了么、商家、送货员
- 我们需要从管理层面来考虑程序的设计
  - 语法仅仅是一个方面
  - 函数是第一步


想象中的程序 VS. 残酷的现实

管理原则
1. 代码重用，减少冗余
2. 逻辑隔离，避免冲突
3. 结构清晰，减少耦合

A brief introduction to Function

A function is a piece of predefined code, which can be called later by other codes

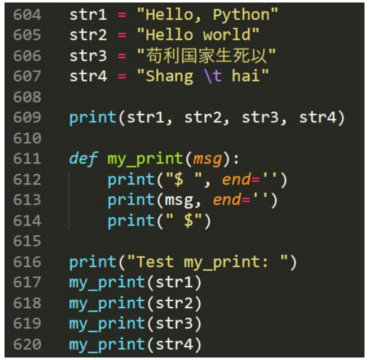
- We have used the following functions:
  - print(“hello world”), print(124), print(123+234)
  - int(“124”), float(“23.45”), str(2123)
  - eval(“1+2+3”)
  - input(“hello world”)
  - type(123), id(‘hello world’)
- The advantage of functions: Reuse, write once and called forever
  - In convenience, we say: invoke a function or call a function (函数调用)
- Function name, parameters and return value (function value)
  - x = int(“1234”), int is the function name, “1234” is the parameter, x is assigned as the return value
  - x = eval(“1+2+3”), eval, “1+2+3”, x
  - x = input(“Please enter a string: ”), input, “Please enter a string: ”, x
  - print(x), print, x, no return value

Define your own function

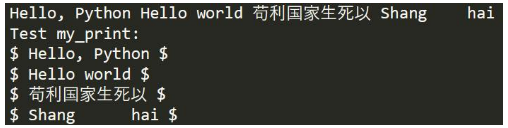
- Three factors of a function: **function name**, **parameters** and **return value**
- Grammar for defining a function
  - def function_name(param1, param2, ….): # param1, param2, …. 参数1， 2， …

        … write your code here….
        … write your code here….
        … write your code here….
        return ……
- def is the keyword for defining functions.
  - It should be at the beginning of the line
- Before each line of your code, you should add one "Tab" for indentation (缩进)
- : at the end of the first line, should not be missed
- The function name is the same with a variable:

  letters, numbers and underscore: _
- return is the keyword for return values to the outside. In some functions, no return. That is, the function do not need to return



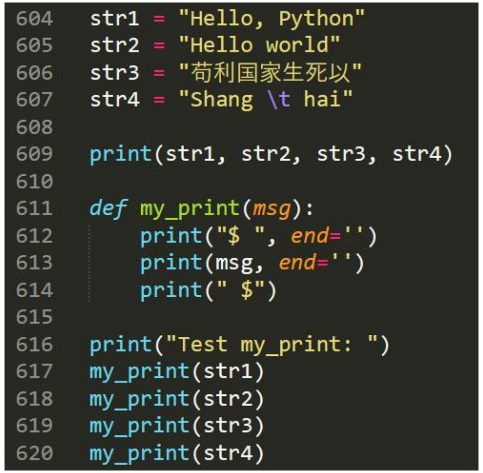
```python
str1 = "Hello, Python"
str2 = "Hello world"
str3 = "苟利国家生死以"
str4 = "Shang \t hai"

print(str1, str2, str3, str4)

def my_print(msg):
    print("$ ", end='')
    print(msg, end='')
    print(" $")

print("Test my_print: ")
my_print(str1)
my_print(str2)
my_print(str3)
my_print(str4)
```



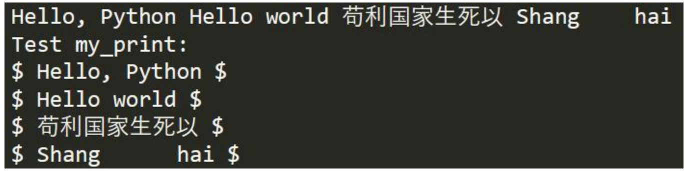
```
Hello, Python Hello world 苟利国家生死以 Shang     hai
Test my_print:
$ Hello, Python $
$ Hello world $
$ 苟利国家生死以 $
$ Shang      hai $
```

函数，内外隔离（空间），一次性（时间），互不干扰

Flow of functions



```
604  str1 = "Hello, Python"
605  str2 = "Hello world"
606  str3 = "苟利国家生死以"
607  str4 = "Shang \t hai"
608
609  print(str1, str2, str3, str4)
610
611  def my_print(msg):
612      print("$ ", end='')
613      print(msg, end='')
614      print(" $")
615
616  print("Test my_print: ")
617  my_print(str1)
618  my_print(str2)
619  my_print(str3)
620  my_print(str4)
```



```
Hello, Python Hello world 苟利国家生死以 Shang    hai
Test my_print:
$ Hello, Python $
$ Hello world $
$ 苟利国家生死以 $
$ Shang      hai $
```

1. 先定义，后运行 # 如果610行， my_print(“error”)?
2. 函数定义后，不会自动被运行，只有被调用的时候才会运行 # 616

从第617行开始，函数运行的一般过程：

1. 从调用函数的地方（617行）开始，跳转到函数定义的开始（611）
2. 参数传递，初始化函数的参数，赋值语句

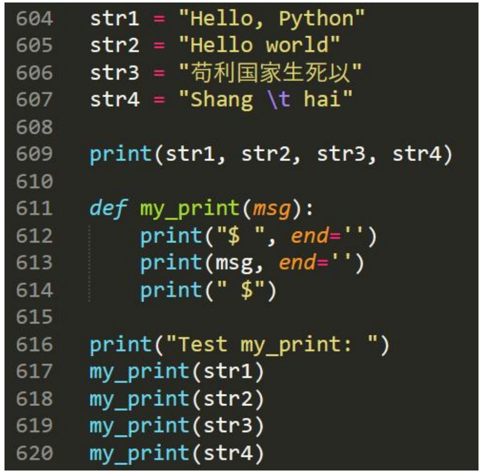
   msg=str1 (611行)
3. 从函数体（612）开始顺序执行
4. 执行完成后（614），返回到调用函数的地方（618），函数内的代码和变量被清除

5. 同理，继续执行618行
6. 同理，继续执行619行
7. 同理，继续执行620行

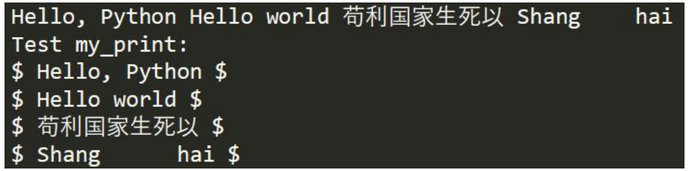
617,618,619,620运行了同样的一段代码，但是互不影响，每次运行后，函数内的代码和数据被清除（一次性）

Indentation缩进

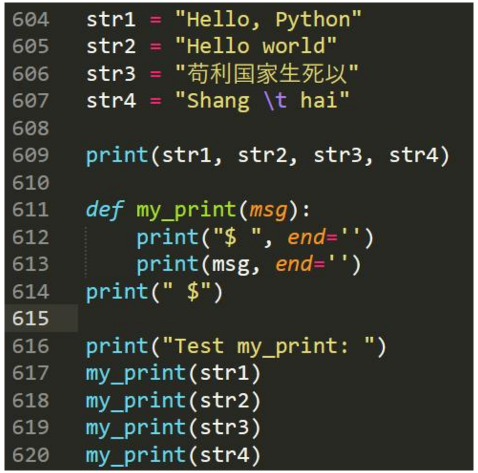
- 连续的具有相同缩进的一段代码，属于同一个代码块。代码块和前面的语句构成逻辑上面的一个整体
  - 英文中，新段落另起一行；同一段落不变



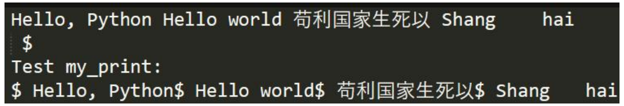
```python
str1 = "Hello, Python"
str2 = "Hello world"
str3 = "苟利国家生死以"
str4 = "Shang \t hai"

print(str1, str2, str3, str4)

def my_print(msg):
    print("$ ", end='')
    print(msg, end='')
    print(" $")

print("Test my_print: ")
my_print(str1)
my_print(str2)
my_print(str3)
my_print(str4)
```



```
Hello, Python Hello world 苟利国家生死以 Shang     hai
Test my_print:
$ Hello, Python $
$ Hello world $
$ 苟利国家生死以 $
$ Shang      hai $
```



```python
str1 = "Hello, Python"
str2 = "Hello world"
str3 = "苟利国家生死以"
str4 = "Shang \t hai"

print(str1, str2, str3, str4)

def my_print(msg):
    print("$ ", end='')
    print(msg, end='')
print(" $")

print("Test my_print: ")
my_print(str1)
my_print(str2)
my_print(str3)
my_print(str4)
```



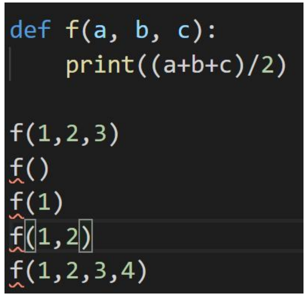
```
Hello, Python Hello world 苟利国家生死以 Shang     hai
 $
Test my_print:
$ Hello, Python$ Hello world$ 苟利国家生死以$ Shang    hai
```

对比体会：

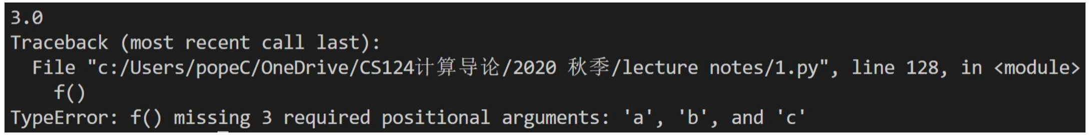
print(" $") 的缩进和前面的语句不一样，已经不属于函数定义范围的语句了

Parameter passing (参数传递)

- To invoke a function, we need to pass parameters to the function
- Take the function $f(x_1, x_2, \ldots, x_n)$ for example
  - When we call f, we need to pass exactly $\boldsymbol{n}$ parameters to f: $f(y_1, y_2, \ldots, y_n)$
  - The types of x_i and y_i should be the same
- Python choose the proper function the given function name and the parameter list



```python
def f(a, b, c):
    print((a+b+c)/2)

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



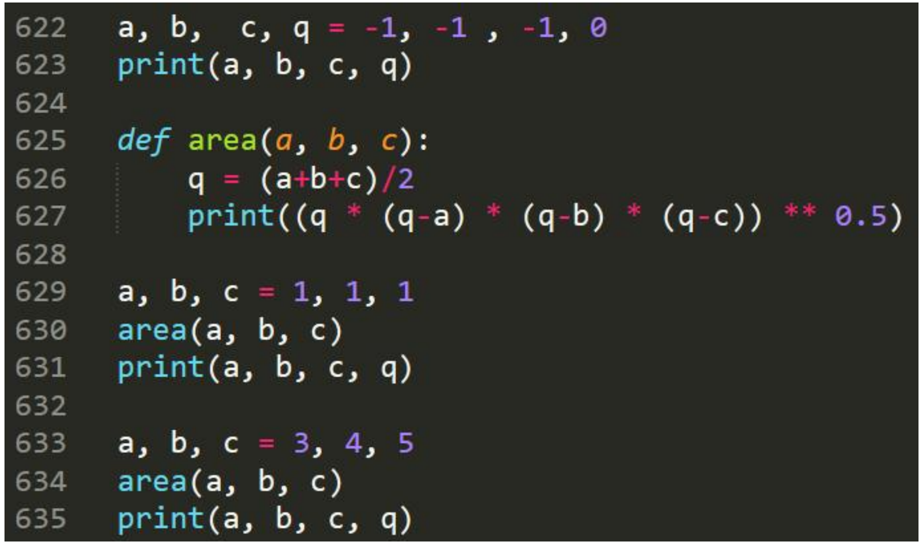
```
3.0
Traceback (most recent call last):
  File "c:/Users/popeC/OneDrive/CS124计算导论/2020 秋季/lecture notes/1.py", line 128, in <module>
    f()
TypeError: f() missing 3 required positional arguments: 'a', 'b', and 'c'
```

参数数量要一致，对齐
参数传递，赋值语句

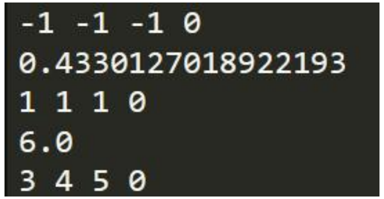
$$x1,\ x2,\ \ldots,\ xn$$
$$=\ y1,\ y2,\ \ldots,\ yn$$

Function isolation

函数内的变量（局部变量）与外部变量（全局变量）不会互相干扰，可以同名 (避免相互冲突)



```python
a, b,  c, q = -1, -1 , -1, 0
print(a, b, c, q)

def area(a, b, c):
    q = (a+b+c)/2
    print((q * (q-a) * (q-b) * (q-c)) ** 0.5)

a, b, c = 1, 1, 1
area(a, b, c)
print(a, b, c, q)

a, b, c = 3, 4, 5
area(a, b, c)
print(a, b, c, q)
```



```
-1 -1 -1 0
0.4330127018922193
1 1 1 0
6.0
3 4 5 0
```

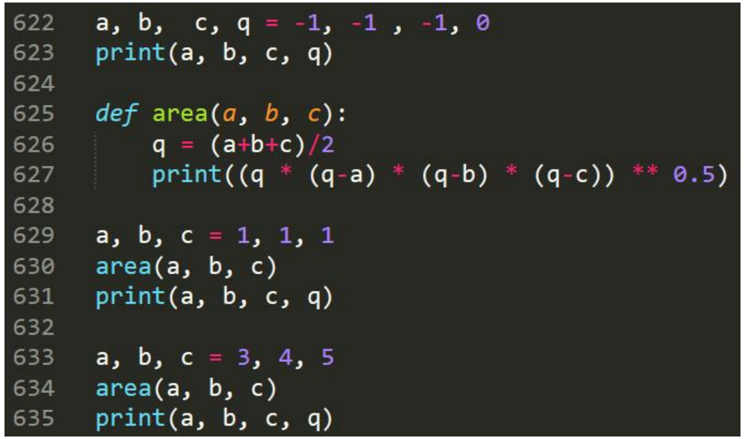
函数，内外隔离（空间），一次性（时间），互不干扰

Function with return value

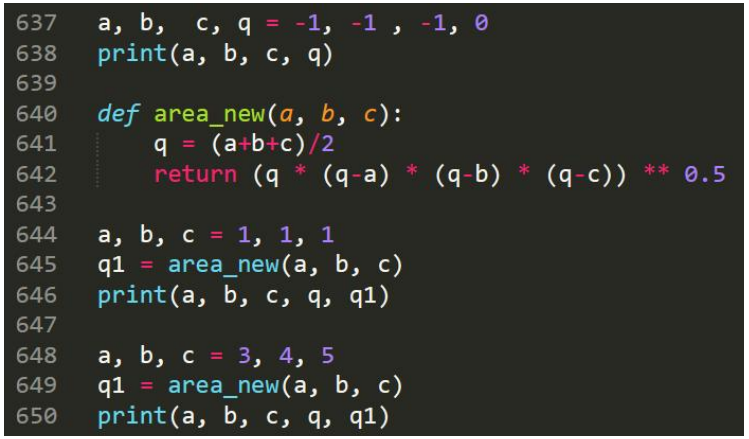
- 函数是一次性的，运行结束后，自动销毁。如何将函数内的值x，传递给函数外？

return x

- 返回值x就是函数值，return是系统保留关键字
- 函数的返回值x可以当作一个变量使用

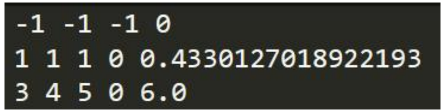


```python
a, b,  c, q = -1, -1 , -1, 0
print(a, b, c, q)

def area(a, b, c):
    q = (a+b+c)/2
    print((q * (q-a) * (q-b) * (q-c)) ** 0.5)

a, b, c = 1, 1, 1
area(a, b, c)
print(a, b, c, q)

a, b, c = 3, 4, 5
area(a, b, c)
print(a, b, c, q)
```



```python
a, b,  c, q = -1, -1 , -1, 0
print(a, b, c, q)

def area_new(a, b, c):
    q = (a+b+c)/2
    return (q * (q-a) * (q-b) * (q-c)) ** 0.5

a, b, c = 1, 1, 1
q1 = area_new(a, b, c)
print(a, b, c, q, q1)

a, b, c = 3, 4, 5
q1 = area_new(a, b, c)
print(a, b, c, q, q1)
```

print -- return
q不变，q1为函数值

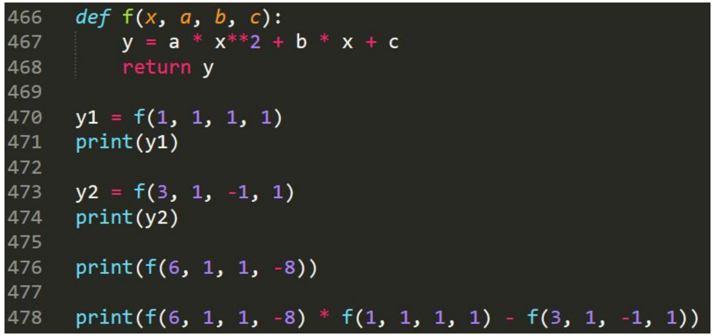


```
-1 -1 -1 0
1 1 1 0 0.4330127018922193
3 4 5 0 6.0
```

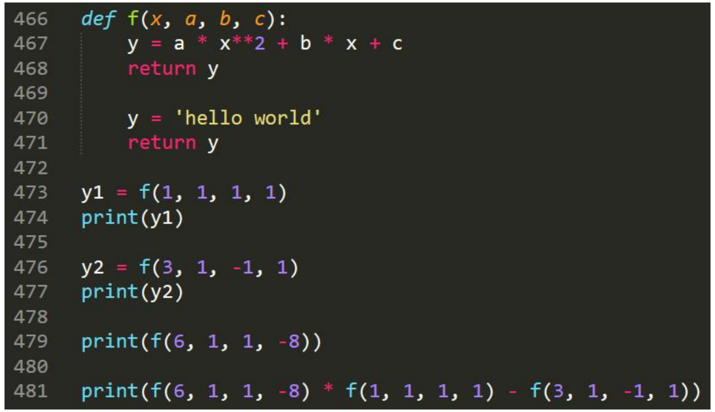
函数，内外隔离（空间），一次性（时间），互不干扰

Return: examples

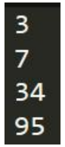
return: only return once



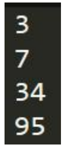
```python
def f(x, a, b, c):
    y = a * x**2 + b * x + c
    return y

y1 = f(1, 1, 1, 1)
print(y1)

y2 = f(3, 1, -1, 1)
print(y2)

print(f(6, 1, 1, -8))

print(f(6, 1, 1, -8) * f(1, 1, 1, 1) - f(3, 1, -1, 1))
```



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

    y = 'hello world'
    return y

y1 = f(1, 1, 1, 1)
print(y1)

y2 = f(3, 1, -1, 1)
print(y2)

print(f(6, 1, 1, -8))

print(f(6, 1, 1, -8) * f(1, 1, 1, 1) - f(3, 1, -1, 1))
```



```
3
7
34
95
```

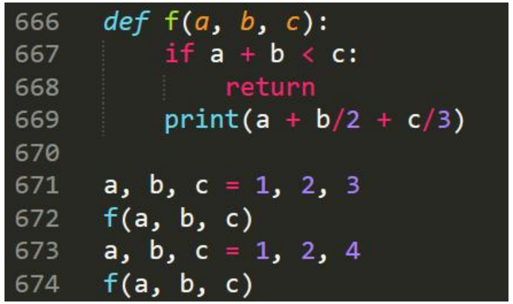


```
3
7
34
95
```

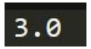
Return: 返回，后面的指令不会执行

return

- return关键字有两个层面的意思：
  - 返回运算结果给函数调用的地方, 然后结束函数运行
    - y=f(x, a, b, c)
  - 可以不带返回值，直接结束函数运行，回到函数调用的地方
    - return用来结束一个函数的运行（函数中只有一个return起作用）
    - 注：break，continue只能结束一个循环的运行 (后续学习)



```python
def f(a, b, c):
    if a + b < c:
        return
    print(a + b/2 + c/3)

a, b, c = 1, 2, 3
f(a, b, c)
a, b, c = 1, 2, 4
f(a, b, c)
```

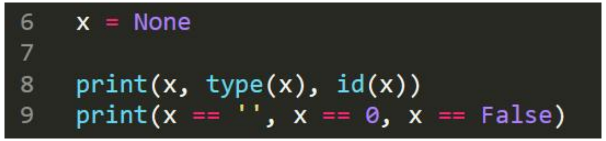


```
3.0
```

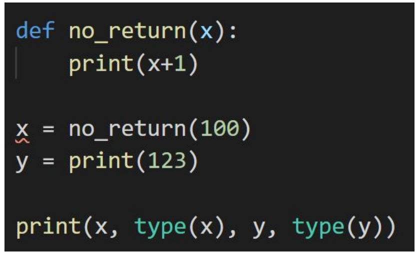
Function without return value

The None keyword is used to define a null value, or no value at all. None is a data type of its own (NoneType) and only None can be None. None is not the same as 0, False, or an empty string.

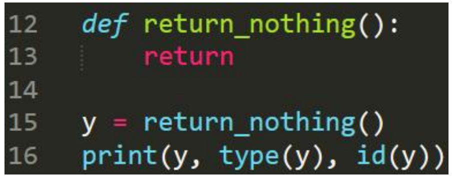
- Functions without return value returns None



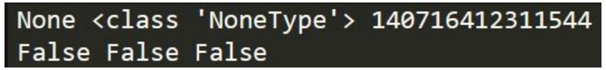
```python
x = None

print(x, type(x), id(x))
print(x == '', x == 0, x == False)
```



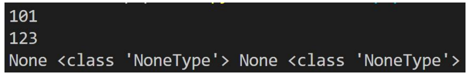
```python
def no_return(x):
    print(x+1)

x = no_return(100)
y = print(123)

print(x, type(x), y, type(y))
```



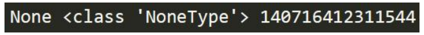
```python
def return_nothing():
    return

y = return_nothing()
print(y, type(y), id(y))
```



```
None <class 'NoneType'> 140716412311544
False False False
```



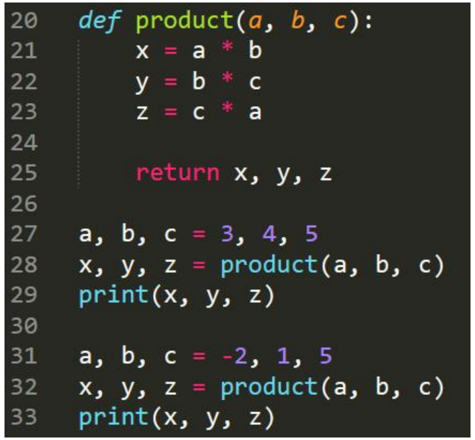
```
101
123
None <class 'NoneType'> None <class 'NoneType'>
```



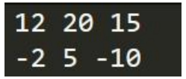
```
None <class 'NoneType'> 140716412311544
```

Return Multiple Values

return x, y, z, w



```python
def product(a, b, c):
    x = a * b
    y = b * c
    z = c * a

    return x, y, z

a, b, c = 3, 4, 5
x, y, z = product(a, b, c)
print(x, y, z)

a, b, c = -2, 1, 5
x, y, z = product(a, b, c)
print(x, y, z)
```



```
12 20 15
-2 5 -10
```

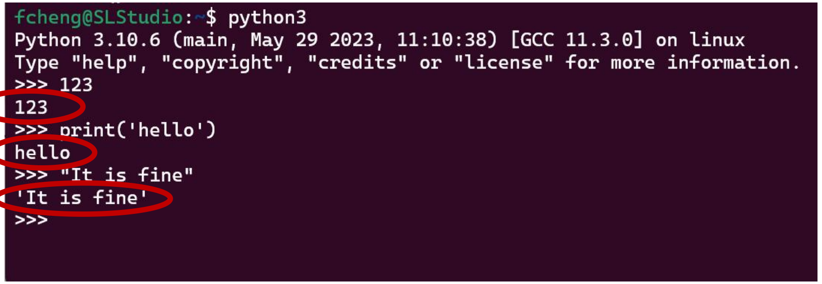
两边变量的数量必须一致

print() VS. return

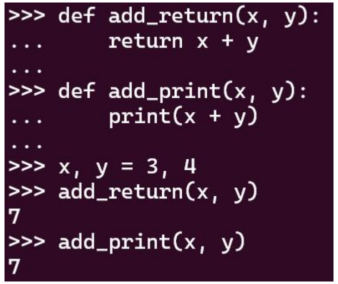
print() 和return没有任何关系。一个是打印，一个是返回

- print()表示打印、输出。在terminal（终端）上输出你希望的内容
- return表示从函数里面返回一个值。调用函数后，你会获得一个值。如果函数定义里面return的时候没有返回值，那就是None
- 在Python自带的解释器中，由于它会自动输出每个表达式的值，所以print(f(1,2,3)) 和f(1, 2, 3)看起来有同样的效果，这是Python自带解释器的额外定义的行为，不属于Python语法的定义。所以在规范的IDE中，譬如Pycharm，是不会有这种效果的。 函数调用，只会获得一个值(可以为None)。

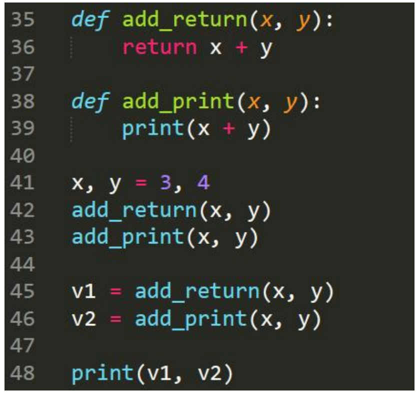
Console的额外行为，造成错觉



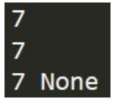
```
fcheng@SLStudio:~$ python3
Python 3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 123
123
>>> print('hello')
hello
>>> "It is fine"
'It is fine'
>>>
```



```
>>> def add_return(x, y):
...     return x + y
...
>>> def add_print(x, y):
...     print(x + y)
...
>>> x, y = 3, 4
>>> add_return(x, y)
7
>>> add_print(x, y)
7
```



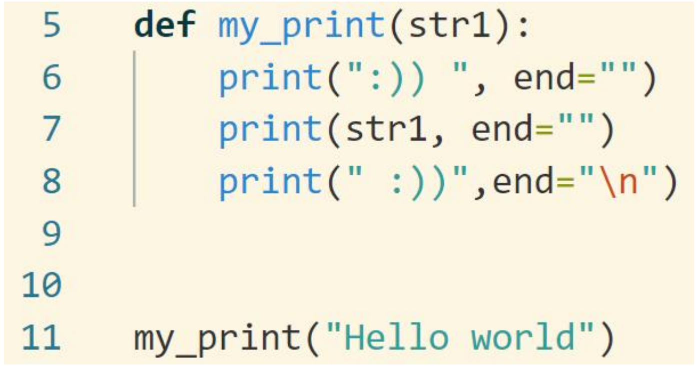
```
def add_return(x, y):
    return x + y

def add_print(x, y):
    print(x + y)

x, y = 3, 4
add_return(x, y)
add_print(x, y)

v1 = add_return(x, y)
v2 = add_print(x, y)

print(v1, v2)
```



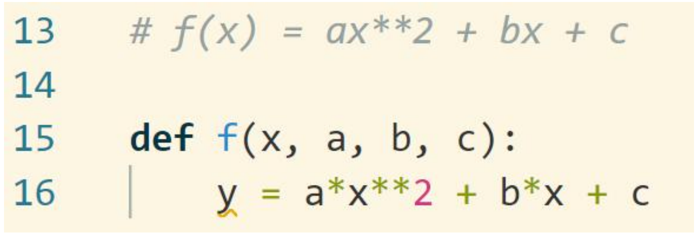
```
7
7
7 None
```

规范：函数定义中，不要用print，多用return，除非问题要求print。Debug时可以用print输出辅助信息

return详解 (1)



```
5   def my_print(str1):
6       print(":)) ", end="")
7       print(str1, end="")
8       print(" :))",end="\n")
9
10
11  my_print("Hello world")
```



```
13  # f(x) = ax**2 + bx + c
14
15  def f(x, a, b, c):
16      y = a*x**2 + b*x + c
```

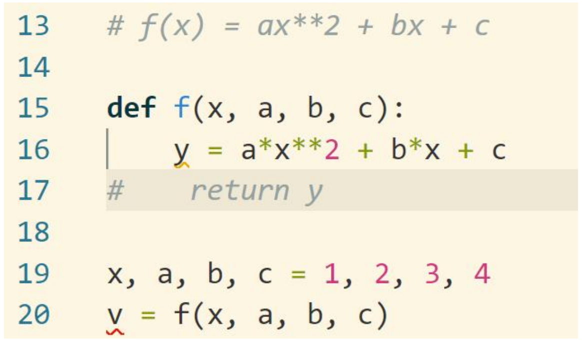
函数运行的基本规律

- 在函数调用中，程序执行的顺序会跳转到被调用函数中，然后依次运行被调用函数中的语句。
- 函数中的语句执行结束后，执行的顺序就回到调用的地方接着继续执行。
- 左边的程序，在运行到第11行时，程序跳转到第5行运行，依次运行到第8行，运行完后，回到第12行。
- 由于函数内部和外部互相不干扰，所以函数运行结束后，里面的变量和数据都”销毁”了

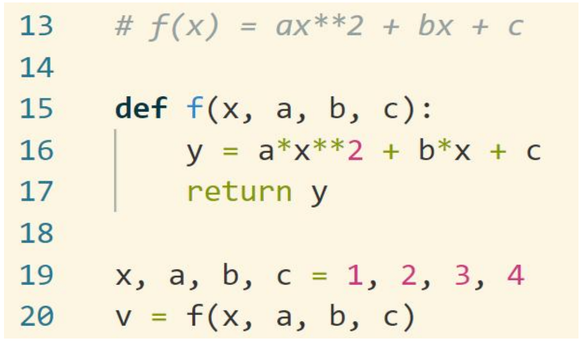
新的问题

- 假设我们需要定义函数$f(x) = ax^2 + bx + c$，一个具有函数值的函数 (例如math.sin(123))
- 按照函数的定义，我们写好了 15，16行的代码
- 我们如何让调用函数的人知道函数运行结果就是y呢？？
- 假定f(x)只有15，16两行，那么f(x)调用结束后，调用者如何自动知道函数值就是y呢？？？
- print(y)??? print()只是打印到控制台，是给人看的，程序并不会自动去看打印的结果。不可行!!!!!!!!
- 我们需要一个新的语法(机制)，来告诉函数调用的人，这个函数的函数值是什么。return !!!!!
- 当然了，函数也可以没有函数值，例如my_print()

return详解 (2)



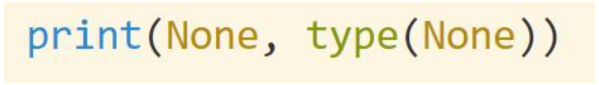
```
13  # f(x) = ax**2 + bx + c
14
15  def f(x, a, b, c):
16      y = a*x**2 + b*x + c
17  #    return y
18
19  x, a, b, c = 1, 2, 3, 4
20  v = f(x, a, b, c)
```



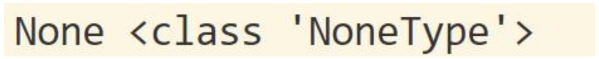
```
13  # f(x) = ax**2 + bx + c
14
15  def f(x, a, b, c):
16      y = a*x**2 + b*x + c
17      return y
18
19  x, a, b, c = 1, 2, 3, 4
20  v = f(x, a, b, c)
```

return

- 上面两个代码，左边没有return y，右边有return y
- 同样对于20行的f()函数调用, 左边的代码运行到16就返回了，不会告诉调用f的用户，函数值是y
- 右边的函数，通过return y命令告诉了用户，函数值为y
- 这样20行，右边就可以把y赋值给v了
- 默认情况下，函数执行结束后，会返回到调用它的地方。如果没有返回值，可以默认返回None，即return None。None在python中表示不存在的意思
- 我们说没有返回值，等价于返回None
- Return vs print()
  - return是告诉调用者，函数的值是多少
  - print是将信息输出到控制台，二者风马牛不相及

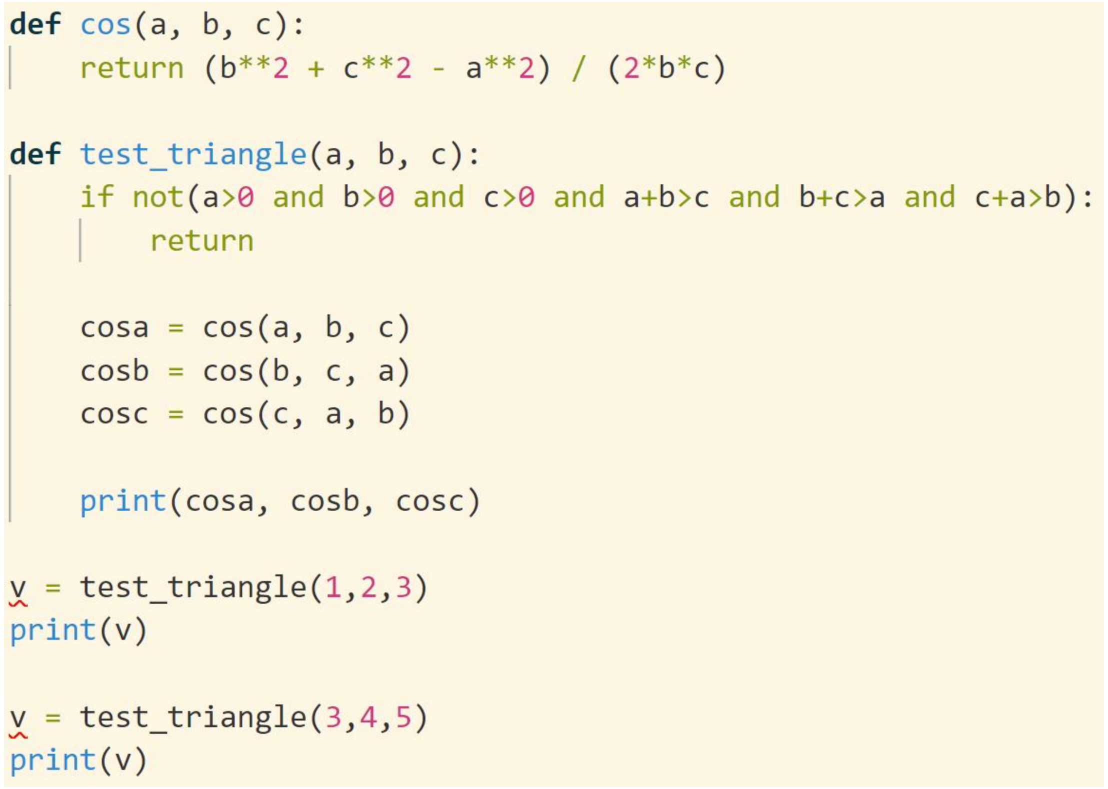


```
print(None, type(None))
```

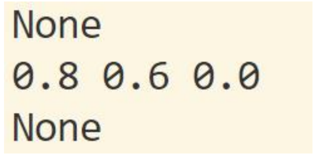


```
None <class 'NoneType'>
```

return详解 (3)

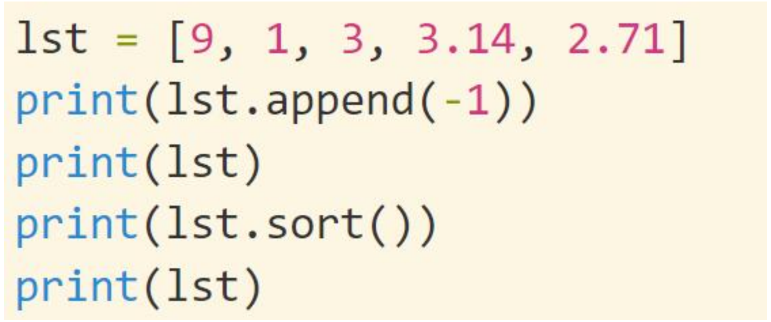


```python
def cos(a, b, c):
    return (b**2 + c**2 - a**2) / (2*b*c)

def test_triangle(a, b, c):
    if not(a>0 and b>0 and c>0 and a+b>c and b+c>a and c+a>b):
        return

    cosa = cos(a, b, c)
    cosb = cos(b, c, a)
    cosc = cos(c, a, b)

    print(cosa, cosb, cosc)

v = test_triangle(1,2,3)
print(v)

v = test_triangle(3,4,5)
print(v)
```

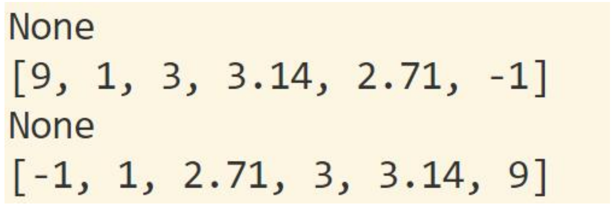


```
None
0.8 0.6 0.0
None
```

return详解 (4)



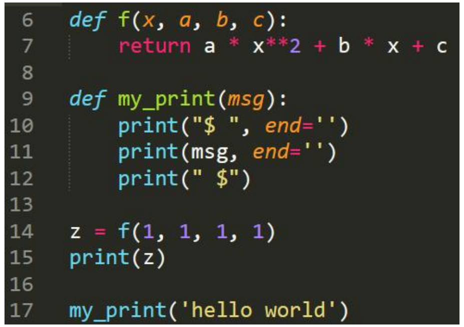
```python
lst = [9, 1, 3, 3.14, 2.71]
print(lst.append(-1))
print(lst)
print(lst.sort())
print(lst)
```



```
None
[9, 1, 3, 3.14, 2.71, -1]
None
[-1, 1, 2.71, 3, 3.14, 9]
```

- Python中的列表在设计时，append()和sort()函数是直接在原列表上修改，不会生成新的列表。所以不需要返回一个新的列表，也就没有返回值(等价于return None)
  - print(lst.append(-1)) 和 print(lst.sort())都会输出None
- 但是lst在append和sort后，都已经发生了改变
- 假定有两个列表a, b，那么c=a+b将生成一个新的列表。生成新的和直接修改会有很多影响

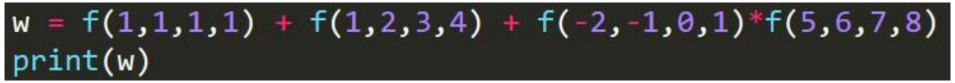
Summary: Function call in Python



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

def my_print(msg):
    print("$ ", end='')
    print(msg, end='')
    print(" $")

z = f(1, 1, 1, 1)
print(z)

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



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

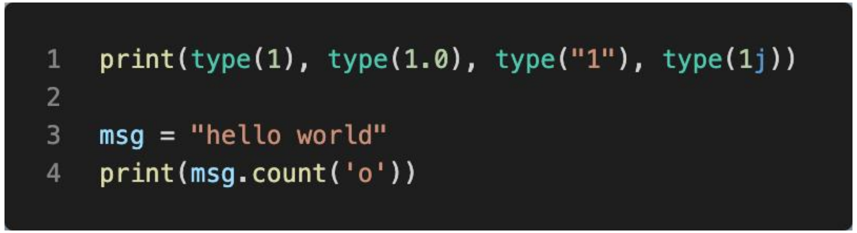
From function to class and module

- The real world is complicated and the software to simulate the real problems will be very large and hence hard to maintain
  - Source lines of code: Windows 2000 (>29M), XP (45M), Vista (60M), Win 8 (50-60M)
- Software development: Reuse, separation
  - Object-oriented programming: C++, Java, Python
  - Functional programming
  - The Mythical Man-Month: 《人月神话：软件项目管理之道》
- In python, we have function, class and module
  - Function: several lines of code
  - Class: data and methods (functions) operated on these data
  - Module: several class that focused on the same field

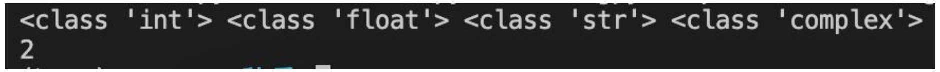
1. https://en.wikipedia.org/wiki/Object-oriented_programming
2. https://en.wikipedia.org/wiki/Functional_programming
3.https://zh.wikipedia.org/wiki/%E4%BA%BA%E6%9C%88%E7%A5%9E%E8%AF%9D

Class

- 数据类型 int, float, complex, str
- Python中，不同类型type()就是不同的class
- class: 把数据和函数打包在一起，就构成一个class。好处之一是：可以和其它的数据和函数隔离开
  - 在三角形研究中，x, y, z 是三条边，可以定义三角形相关的函数
  - 在代数问题中，x, y, z是多项式的变量，可以定义多项式相关的函数
  - 为了避免同一文件中，可能的x, y, z冲突，用class将三角形和多项式分别打包为类，隔离开
- class中的函数和变量属于这个类所特有，不会和外面的同名函数或者变量冲突
- 用法 x.func(parm)
  - . 表示func是x中的函数，不是其它地方的 (先学会用，具体原理在第9讲)



```python
print(type(1), type(1.0), type("1"), type(1j))

msg = "hello world"
print(msg.count('o'))
```



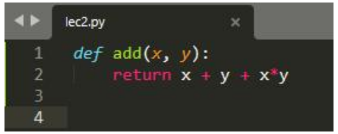
```
<class 'int'> <class 'float'> <class 'str'> <class 'complex'>
2
```

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

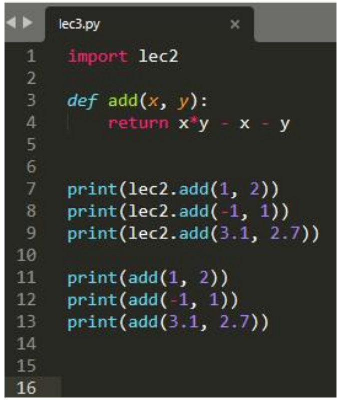
Module

A Python file is called a module. Use a module: import module_name (规范：全小写字母)

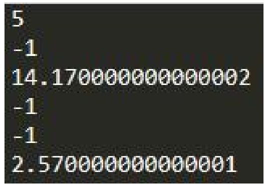
- Suppose we have a Python file lec2.py, which has defined a function called add(x, y).
- In lecture 3, we would like to invoke add(x, y) in lec3.py.
- We also define a new add(x, y) in lec3.py.
- In the future lecture 4, we invoke add() in both lec 2 & 3.


lec2.py

```python
def add(x, y):
    return x + y + x*y

```

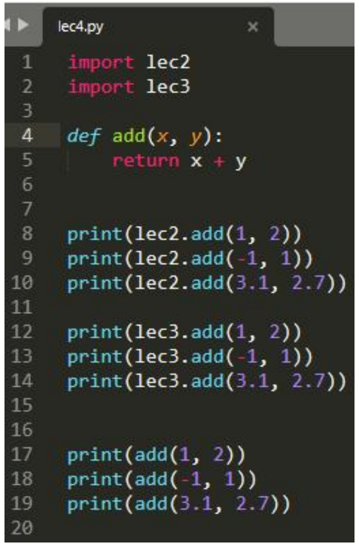

lec3.py

```python
import lec2

def add(x, y):
    return x*y - x - y


print(lec2.add(1, 2))
print(lec2.add(-1, 1))
print(lec2.add(3.1, 2.7))

print(add(1, 2))
print(add(-1, 1))
print(add(3.1, 2.7))


```

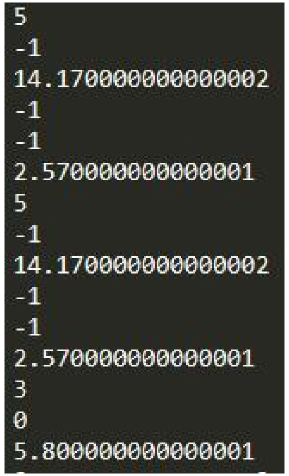


```
5
-1
14.170000000000002
-1
-1
2.570000000000001
```

lec2.py,lec3.py, lec4.py
必须在同一个文件夹


lec4.py

```python
import lec2
import lec3

def add(x, y):
    return x + y


print(lec2.add(1, 2))
print(lec2.add(-1, 1))
print(lec2.add(3.1, 2.7))

print(lec3.add(1, 2))
print(lec3.add(-1, 1))
print(lec3.add(3.1, 2.7))


print(add(1, 2))
print(add(-1, 1))
print(add(3.1, 2.7))

```

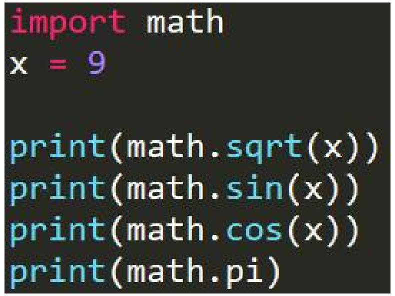


```
5
-1
14.170000000000002
-1
-1
2.570000000000001
5
-1
14.170000000000002
-1
-1
2.570000000000001
3
0
5.800000000000001
```

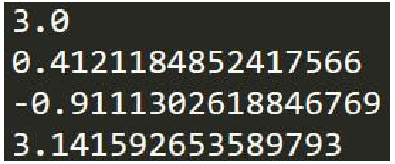
$lec2.add()$: 表明用到了$lec2$模块中的$add()$函数。其它的模块($lec3$)中也可以有$add()$函数。
避免名字冲突：小明
湖北省武汉市 小明
湖南省长沙市 小明

Math Module

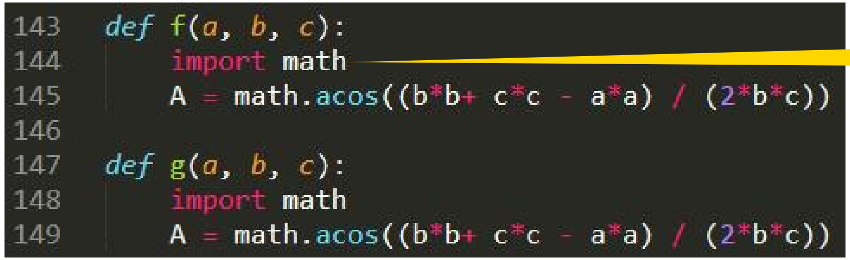
- 数学函数模块: 预先写好的常用数学函数代码
  - 例如sin(), cos(), sqrt(), ceil(), floor(), factorial()
- 用法： **import** math #申明导入math模块 （一般放在文件开始几行）
  - 表明现在要用到math库了



```python
import math
x = 9

print(math.sqrt(x))
print(math.sin(x))
print(math.cos(x))
print(math.pi)
```

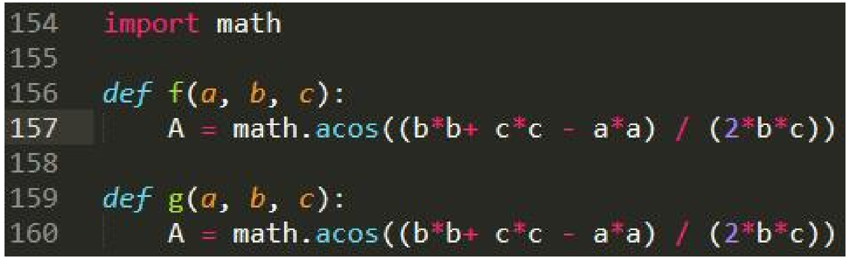


```
3.0
0.4121184852417566
-0.9111302618846769
3.141592653589793
```



```python
def f(a, b, c):
    import math
    A = math.acos((b*b+ c*c - a*a) / (2*b*c))

def g(a, b, c):
    import math
    A = math.acos((b*b+ c*c - a*a) / (2*b*c))
```

不建议



```python
import math

def f(a, b, c):
    A = math.acos((b*b+ c*c - a*a) / (2*b*c))

def g(a, b, c):
    A = math.acos((b*b+ c*c - a*a) / (2*b*c))
```

import一次，文件开始位置

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

How to program?

- Understand the basic grammar well
- Remember the common usage and example
- Practice makes perfect
  - Ask the python compiler for help to answer your questions
    - int(3.1)?
- PEP 8 -- Style Guide for Python Code
  - https://www.python.org/dev/peps/pep-0008/#tabs-or-spaces