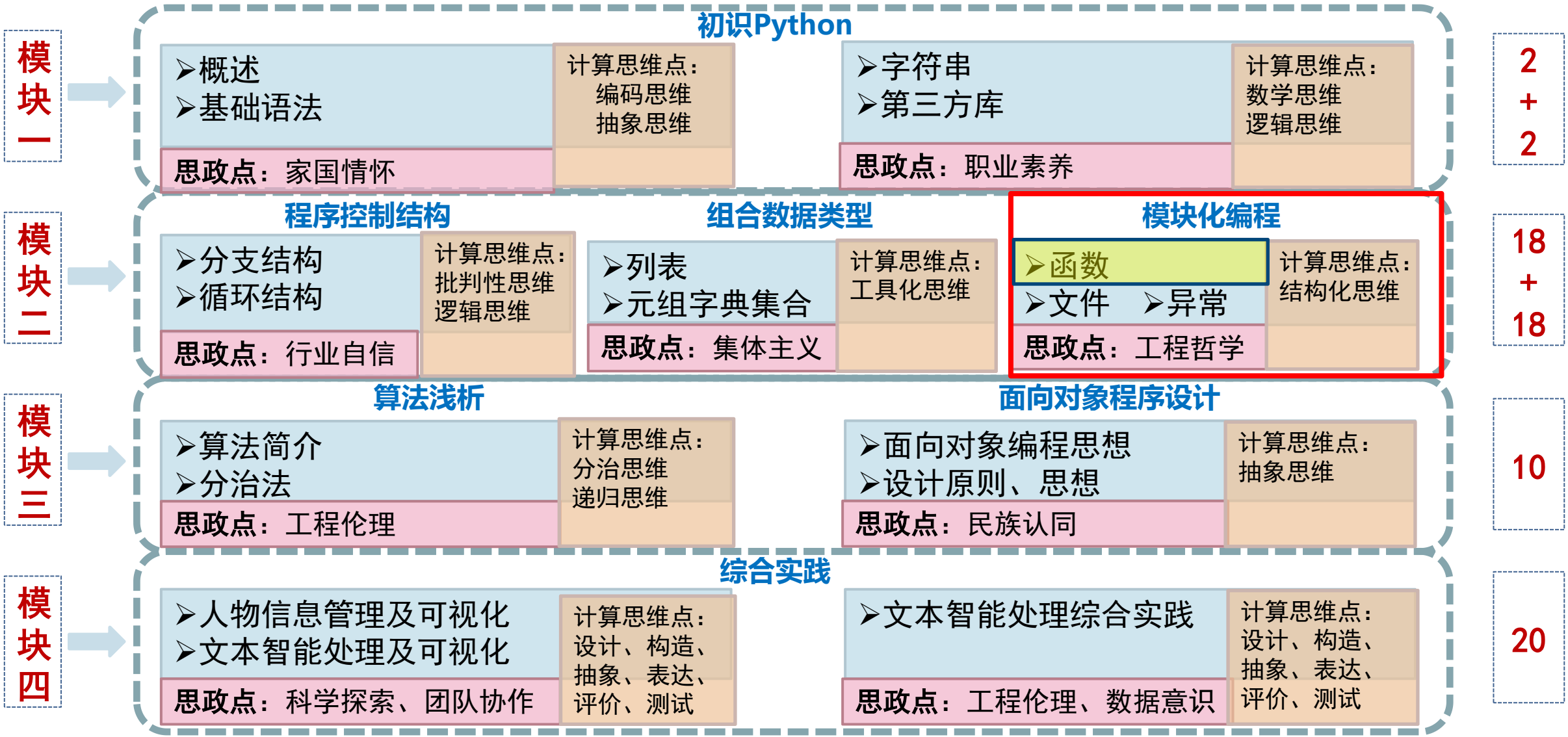

程序设计（Python）

# 函　数

主讲：数据与目标工程学院 胡瑞娟 副教授

# 知识回顾


模块一
初识Python
➢概述
➢基础语法
思政点：家国情怀
计算思维点：
编码思维
抽象思维
➢字符串
➢第三方库
思政点：职业素养
计算思维点：
数学思维
逻辑思维
2
+
2
模块二
程序控制结构
➢分支结构
➢循环结构
思政点：行业自信
计算思维点：
批判性思维
逻辑思维
组合数据类型
➢列表
➢元组字典集合
思政点：集体主义
计算思维点：
工具化思维
模块化编程
➢函数
➢文件
➢异常
思政点：工程哲学
计算思维点：
结构化思维
18
+
18
模块三
算法浅析
➢算法简介
➢分治法
思政点：工程伦理
计算思维点：
分治思维
递归思维
面向对象程序设计
➢面向对象编程思想
➢设计原则、思想
思政点：民族认同
计算思维点：
抽象思维
10
模块四
综合实践
➢人物信息管理及可视化
➢文本智能处理及可视化
思政点：科学探索、团队协作
计算思维点：
设计、构造、
抽象、表达、
评价、测试
➢文本智能处理综合实践
思政点：工程伦理、数据意识
计算思维点：
设计、构造、
抽象、表达、
评价、测试
20

# 案例引入

【案例背景】在军事战争中，不同的作战单位需要配备不同数量的弹药。现代智能化作战背景下，假设有一种新型AI弹药分配系统，可以根据作战单位的类型和任务需求，快速计算所需的弹药数量。在分配弹药时，需要考虑不同弹药类型的排列组合方式对装载方案的影响。

任务：

（1）计算n种弹药的分配方案数：一个作战单位有n种不同类型的弹药，计算这些弹药的所有可能分配顺序。

求n!（写出代码）

（2）从n种弹药中选取m种进行排列的方案数：在自动装弹机中，弹药的顺序可能会影响装填速度，考虑不同弹药的装填顺序。

P(n, m) = n! / (n-m)!

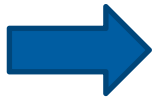
（3）从n种弹药中选取m种进行组合的方案数：计算从弹药库中选择哪些弹药类型，不考虑顺序。

C(n, m) = n!/(m!*(n-m)!)

问题1：计算n种弹药的分配方案数。

**求n!（1*2*3....*n）的值**

```
n=eval(input())
s=1
for i in range(1,n+1):
    s=s*i
print(s)
```



```
def fact(n):
    s=1
    for i in range(1,n+1):
        s=s*i
    return s
```

**问题2、3需要重复编写这些语句**

函数

# 主要内容

01 函数的定义和调用

02 函数的参数传递

03 变量的作用域

01

# 函数的定义与调用

# 1. 函数的定义与调用

- 函数是一段具有**特定功能**的、可**重用**的语句组，用**函数名**来表示并通过函数名完成功能**调用**。
  - 函数可以看作一段具有名字的子程序，可以在需要的地方调用执行，不需要在每个执行地方**重复编写**这些语句。
  - 每次使用函数可以提供**不同**的参数作为**输入**，以实现对不同数据的处理。
  - 函数执行后，反馈相应的**结果**。
  - 使用函数主要有两个目的：**代码重用**和**降低编程难度**。

立正！

队长同志
三班应到8名
实到7名
请指示！
值班员张三

指导员同志
四排应到25名
实到25名
请指示！
值班员李四

## 函数的分类

- Python解释器自带了一些函数和方法，称为**内置的函数**

  （如print（），input（）、eval（））

- Python的**标准库中的函数**（如math库中的sin（））。
- 有些函数是用户自己编写的，称为**自定义函数**。

# 1. 函数的定义与调用

## Python内置函数

查看所有的内置函数（共68个）

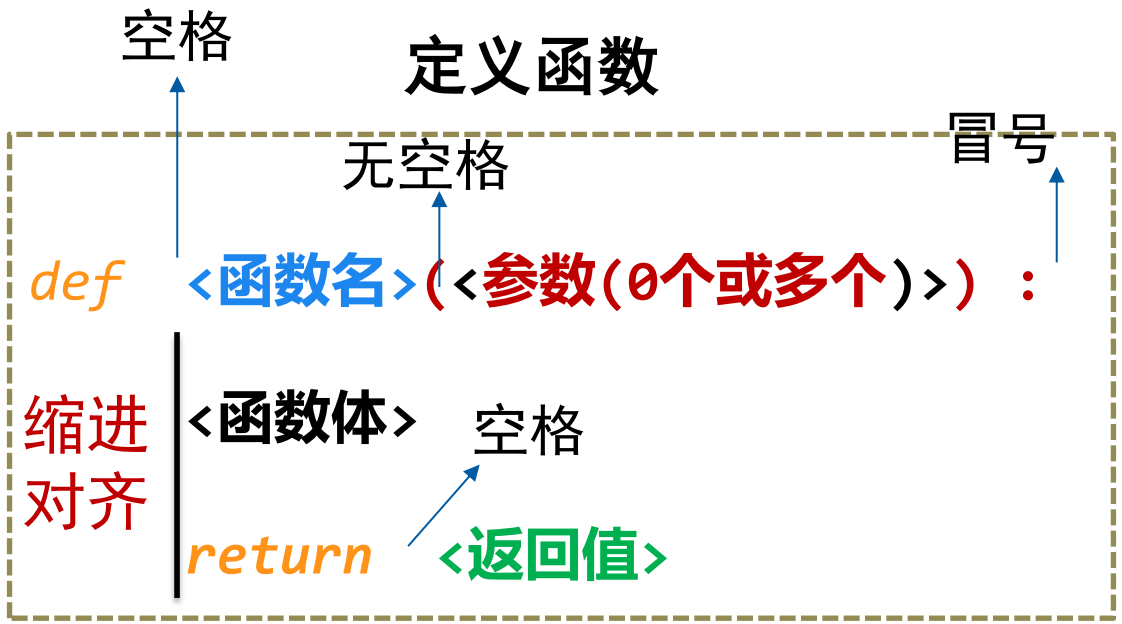
```
import builtins
print(dir(builtins))
```

| | | | | |
|---|---|---|---|---|
| abs() | id() | round() | compile() | locals() |
| all() | input() | set() | dir() | map() |
| any() | int() | sorted() | exec() | memoryview() |
| asci() | len() | str() | enumerate() | next() |
| bin() | list() | tuple() | filter() | object() |
| bool() | max() | type() | format() | property() |
| chr() | min() | zip() | frozenset() | repr() |
| complex() | oct() | | getattr() | setattr() |
| dict() | open() | | globals() | slice() |
| divmod() | ord() | bytes() | hasattr() | staticmethod() |
| eval() | pow() | delattr() | help() | sum() |
| float() | print() | bytearray() | isinstance() | super() |
| hash() | range() | callable() | issubclass() | vars() |
| hex() | reversed() | classmethod() | iter() | import() |

# 1. 函数的定义与调用

## 1.1函数的定义

- Python定义一个函数使用**def**




```
def print_(x):
    if type(x) == float:
        print("%.4f" % x)
    else:
        print(x)
```

**第一次见def是什么时候？**

**函数的四要素：**

（1）函数名：满足标识符的规则

（2）函数参数：放在括号中，括号不能省略

（3）函数体

（4）返回值

# 1. 函数的定义与调用

## 1.1 函数的定义

```
def <函数名>(<参数列表>):
    <函数体>
    return <返回值列表>
```

- **函数名**

函数名可以是任何有效的Python**标识符**。

- 由大小写字母，数字，下划线组成，并且第一个字符必须是字母或者下划线
- 自定义的函数名字不能与Python内置函数名重复

```
get_add    _process    print    AbcDef-123
```

## 1.1 函数的定义

```
def <函数名>(<参数列表>):
    <函数体>
    return <返回值列表>
```

- **参数列表**
  - 函数定义中参数列表里面的参数是**形式参数**，简称为“**形参**”
  - 参数可以有零个、一个或多个，有多个参数则使用逗号隔开
  - 无参数，也必须保留**空括号**

# 1. 函数的定义与调用

## 1.1 函数的定义

```
def <函数名>(<参数列表>):
    <函数体>
    return <返回值列表>
```

- **函数体**

  函数体是函数每次被调用时执行的代码，由一行或多行语句组成。

- **返回值**

  - 如果函数体中不包含return语句，则返回None值

```
def test(a,b):
    s=a+b
```

# 1. 函数的定义与调用

## 1.1函数的定义

```
def <函数名>(<参数列表>):
    <函数体>
    return <返回值列表>
```

### ➢ 返回值

- 如果函数体中包含return语句，则执行到return时则结束函数执行，并返回return后的值（单个，多个）。

```
def test(a,b):
    result=a+b
    return result
    c=a-b
    return c
```

# 1. 函数的定义与调用

## 1.1函数的定义

队长同志
三班应到8名
实到7名
请指示!
值班员张三

指导员同志
四排应到25名
实到25名
请指示!
值班员李四

```
def report(lname, f, allnum, num, name):
    print("{}同志".format(lname))
    print("{}应到{}名".format(f,allnum))
    print("实到{}名".format(num))
    print("请指示!")
    print("值班员{}".format(name))
```

## 1.1函数的定义

【举一反三】定义一个函数isPal，根据输入判断该输入字符串是否为一个回文字符串，返回布尔值。

```
def isPal(x):
    x2 = x[::-1]
    return x==x2
```

# 1. 函数的定义与调用

【案例分析】 问题1：计算n种弹药的分配方案数。用函数解决?

定义函数n!

```
n=eval(input())
s=1
for i in range(1,n+1):
    s=s*i
print(s)
```

```
def fact(n):
    s=1
    for i in range(1,n+1):
        s=s*i
    return s
```

【强调】从设计者（实现函数的人）角度来看：

（1）参数：要处理的对象（I）

（2）函数体：处理过程（P）

（3）返回值：要得到的值（O）

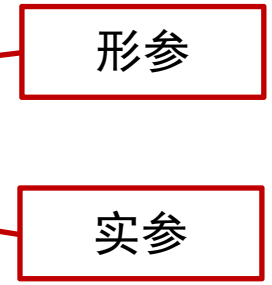
## 1.2 函数的调用

- 函数调用和执行的一般**形式**是：

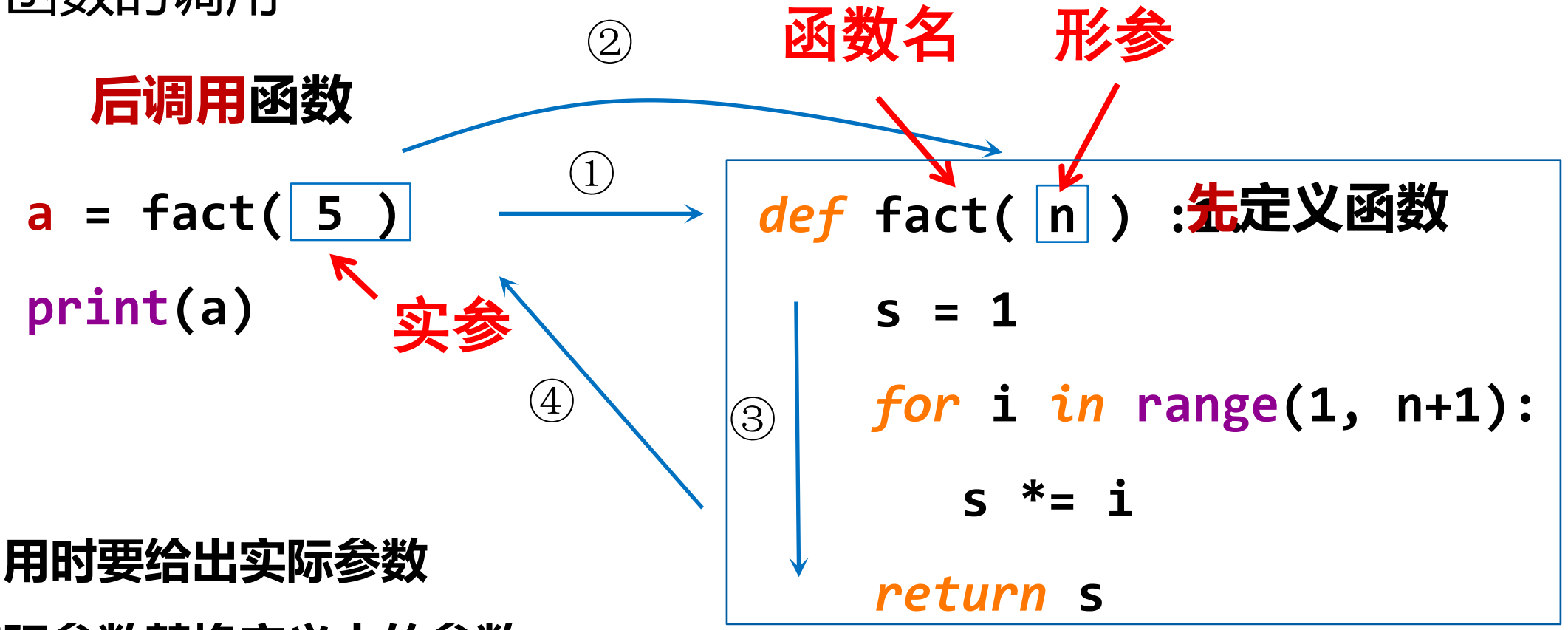
  **<函数名>(<参数列表>)**

- 参数列表中给出要传进入函数内部的参数，称为**实际参数**，简称为“**实参**”。

```
def isPal(x):
    print(x)
isPal('abba')
```



## 1.2 函数的调用




- 调用时要给出实际参数
- 实际参数替换定义中的参数
- 函数调用后得到返回值

返回值

## 1.2 函数的调用（多个参数）

首长同志
五队应到101名
实到99名
请指示！
值班员王五

```
def report(lname, f, allnum, num, name):
    print("{}同志".format(lname))
    print("{}应到{}名".format(f,allnum))
    print("实到{}名".format(num))
    print("请指示!")
    print("值班员{}".format(name))
```

**写出调用函数?**

report("首长","五队",101,99,"王五")

## 【举一反三】编写程序为Mike和Lily输出生日歌（使用函数happyB）

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Mike!
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lily!
Happy birthday to you!
```

```
def happyB(name):
    print("Happy birthday to you!")
    print("Happy birthday to you!")
    print("Happy bithday, dear {}!".format(name))
    print("Happy birthday to you!")
【补充代码】
```

## 【举一反三】编写程序为Mike和Lily输出生日歌（使用函数happyB）

```
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Mike!
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lily!
Happy birthday to you!
```

```
def happy():
  print("Happy birthday to you!")
def happyB(name):
    happy()
    happy()
    print("Happy bithday, dear {}!".format(name))
    happy()
```

**你知道这个程序是如何执行的吗?**

## 1.2 函数的调用

- 程序调用一个函数需要**执行**以下四个步骤（**执行过程**）：

（1）调用程序在调用处**暂停**执行；

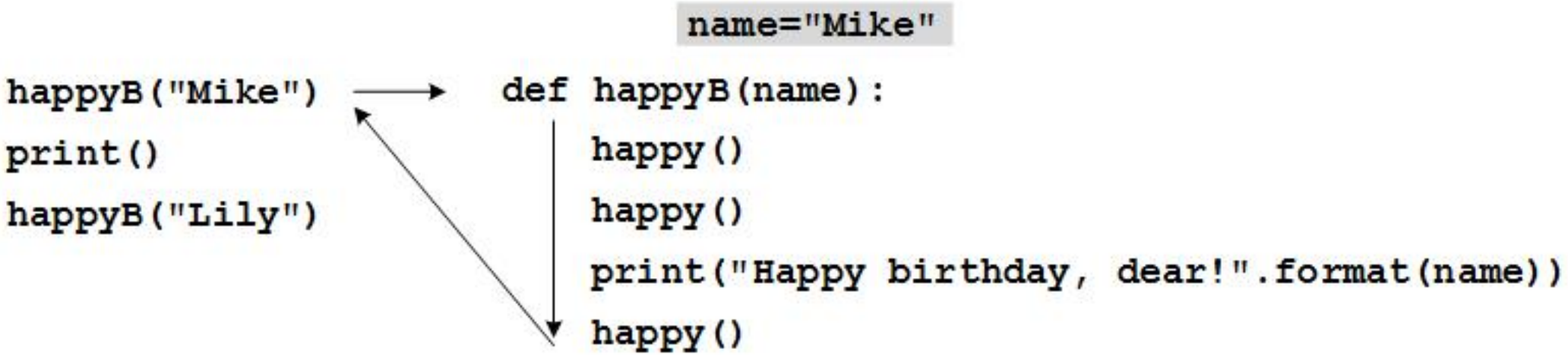
（2）在调用时将实参**传递**给函数的形参；

（3）**执行函数体**语句；

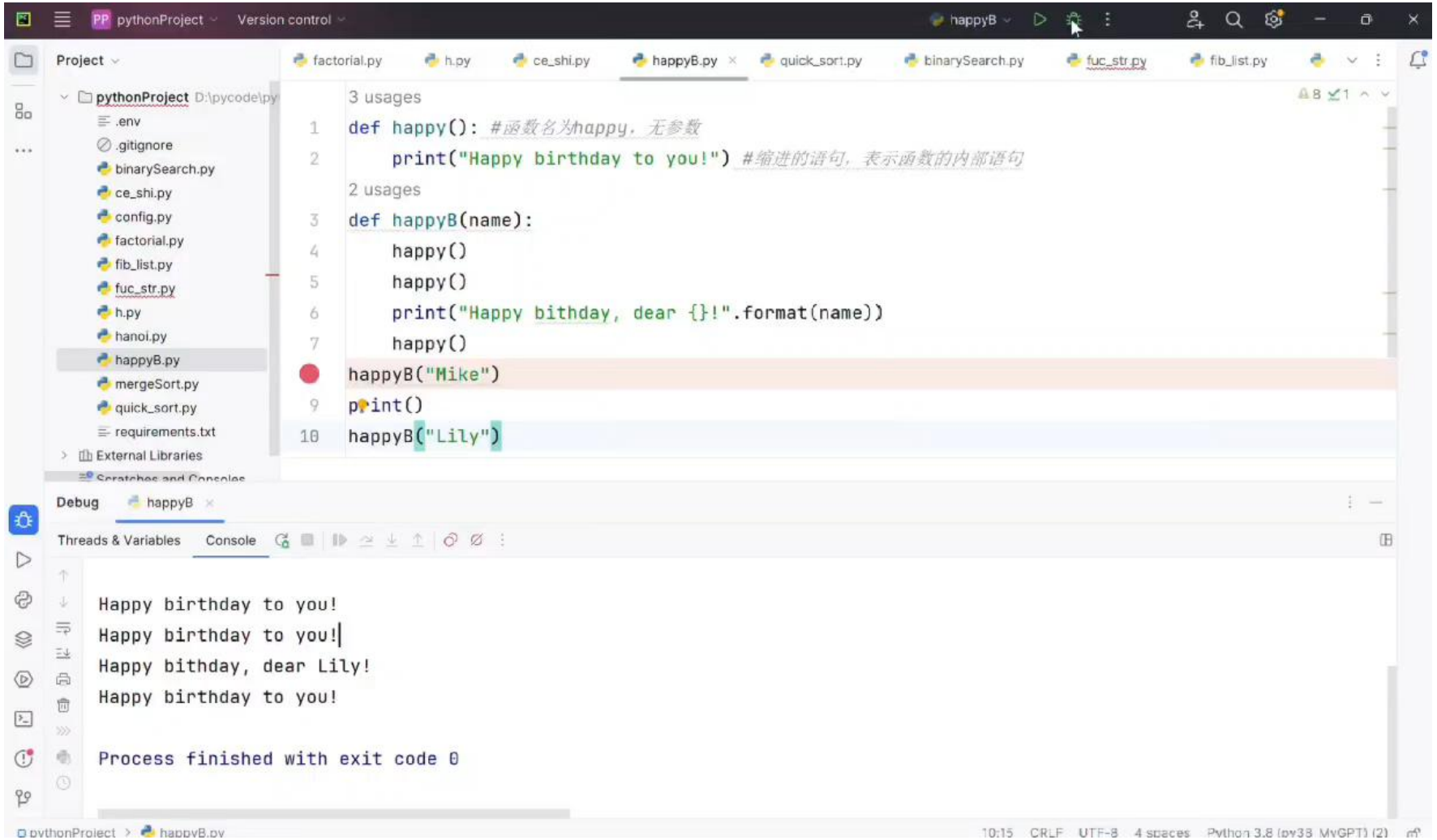
（4）函数调用结束给出返回值，程序**回到调用前的暂停处继续**执行。

## 1.2 函数的调用

```
def happy():
    print("Happy birthday to you!")
def happyB(name):
    happy()
    happy()
    print("Happy birthday, dear {}!".format(name))
    happy()
happyB("Mike")
print()
happyB("Lily")
```



# 1. 函数的定义与调用


pythonProject
Version control
happyB
Project
factorial.py
h.py
ce_shi.py
happyB.py
quick_sort.py
binarySearch.py
fuc_str.py
fib_list.py
pythonProject D:\pycode\py
.env
.gitignore
binarySearch.py
ce_shi.py
config.py
factorial.py
fib_list.py
fuc_str.py
h.py
hanoi.py
happyB.py
mergeSort.py
quick_sort.py
requirements.txt
External Libraries
3 usages
def happy(): #函数名为happy，无参数
print("Happy birthday to you!") #缩进的语句，表示函数的内部语句
2 usages
def happyB(name):
happy()
happy()
print("Happy bithday, dear {}!".format(name))
happy()
happyB("Mike")
print()
happyB("Lily")
Debug
happyB
Threads & Variables
Console
Happy birthday to you!
Happy birthday to you!
Happy bithday, dear Lily!
Happy birthday to you!
Process finished with exit code 0

# 1. 函数的定义与调用

## 【案例分析】

问题2：从n种弹药中选取m种进行排列的方案数。

问题3：从n种弹药中选取m种进行组合的方案数。

$$P(n, m) = n! / (n-m)!$$

$$C(n, m) = n!/(m!*(n-m)!)$$

```
# 定义排列函数
def perm(n, m):
    """计算从n个元素中取m个元素的排列数"""
    if m > n:
        return 0
    per_nm= fact(n)//fact(n-m)
    return per_nm
```

```
# 定义组合函数
def comb(n, m):
    """计算从n个元素中取m个元素的组合数"""
    if m > n:
        return 0
    comb_nm = fact(n)//(fact(m)*fact(n-m))
    return comb_nm
```

```
# 调用函数
fac = fact(5)
per = perm(5,3)
com = comb(5,3)
print("5种弹药的分配方案数:", fac)
print("从5种弹药中选3种进行排列的方案数:", per)
print("从5种弹药中选3种的组合数:", com)
```

02

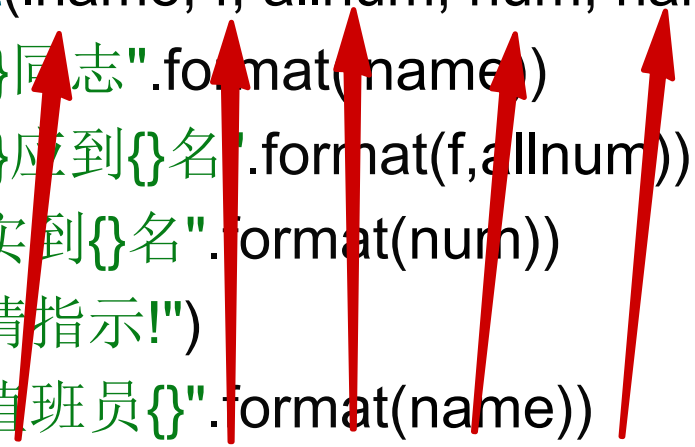
# 函数的参数传递

# 2. 函数的参数传递

## 2.1 位置传参

- 在传参的时候，实参传递的顺序按照形参定义的顺序进行传递的传参方式。

```
def report(lname, f, allnum, num, name):
    print("{}同志".format(name))
    print("{}应到{}名".format(f,allnum))
    print("实到{}名".format(num))
    print("请指示!")
    print("值班员{}".format(name))
report("首长","五队",101,99,"王五")
```



第一个实参给第一个形参赋值，第二个实参给第二个形参赋值，像这种按位置传递的方式就是位置传参

## 2.1 位置传参

- 在传参的时候，需要注意参数的顺序和数量必须与形参保持一致。

```
def index2(x,y):
    return x[y]

index2(2,'abc')
```

```
TypeError: 'int' object is not subscriptable
```

```
def pow2(x,y):
    return x**y

pow2(123)
```

```
TypeError: pow2() missing 1 required positional argument: 'y'
```

## 【举一反三】分析程序运行结果。

```
def func1(str1, str2):
    if str1 > str2:
        resultStr = str1[1:]
    else:
        resultStr = str2[:-1]
    return resultStr
str1 = input("Input a string:")
str2 = input("Input a second string:")
print(func1(str1, str2))
print(func1(str2, str1))
```

输入分别为abc123和bcd456，两个print分别输出？

bcd45
cd456

输入分别为aaabbc和aaabbccd，两个print分别输出？

aaabbcc
aabbccd

## 2.2 关键字传参（名称传参）

- 当参数很多时，一般的调用参数方式**可读性较差**。
  - 假设func()函数有6个参数，分别表示2组三维坐标值。

```
def func(x1, y1, z1, x2, y2, z2):
```

  - 它的一个实际调用如下：

```
result = func(1, 2, 3, 4, 5, 6)
```

  - 如果仅看实际调用而不查看函数定义，很难理解这些输入参数的含义。

## 2.2 关键字传参（名称传参）

- 为了解决上述问题，Python提供了按照形参名称输入实参的方式，函数调用如下：

```
result = func(x2=4, y2=5, z2=6, x1=1, y1=2, z1=3)
```

  - 可以忽略顺序
  - 关键字只能是形参中声明过的才能使用
  - 位置参数和关键字参数混用，必须位置参数在前，关键字参数在后，才能正常使用

```
result = func(1, 2, x2=4, y2=5, z2=6, z1=3)
```

## 2.3 默认参数

- 在定义函数时，如果有些形参存在默认值，可以在定义函数时直接给这些参数指定默认值。
  - 当调用函数的时候，如果不给默认参数赋值，函数就会使用默认值。如果给默认函数参数赋值，函数就使用重新赋的值。
  - 默认参数一定要写到形参的后面，否则将报错。

```
    def Sum(b=3,a,c=5):
              ^
SyntaxError: non-default argument follows default argument
```

# 2. 函数的参数传递

## 2.3 默认参数

为参数指定默认值，称为可选参数

计算 n!

```
def fact(n=5) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
```

调用函数　y=fact()

运行结果　120

调用函数　y=fact(6)

运行结果　720

# 2. 函数的参数传递

## 【举一反三】m到n(包含n)累乘，m默认为1。

函数定义

```
def acc(n, m=1):
    s=1
    for i in range(m, n+1 ):
        s=s*i
    return s
```

可选参数放后面

| 函数调用 | 运行结果 |
| --- | --- |
| `z=acc(5)`<br>`print(z)` | 120 |

位置传参

| 函数调用 | 运行结果 |
| --- | --- |
| `z=acc(5,2)`<br>`print(z)` | 120 |

位置传参

| 函数调用 | 运行结果 |
| --- | --- |
| `z=acc(m=2,n=5)`<br>`print(z)` | 120 |

名称传参

**【举一反三】分析程序运行结果。**

```
def greet(name, greeting='Hello'):
    print(f'{greeting}, {name}!')
```

#可选参数

```
greet('Alice')
greet('Bob', 'Hi there')
greet('Charlie', greeting='Howdy')
```

```
Hello, Alice
Hi there, Bob
Howdy, Charlie
```

```
greet(greeting='Howdy', 'Charlie’)
```

×

# 名称传递必须在位置参数之后，导致语法错误

**【举一反三】判断对错。**

（1）一个函数只能有一个参数。

（2）函数都有返回值。

（3）在使用相同函数时，每次传入参数的个数是相同的。

A　对　错　对

B　错　对　对

C　错　错　错

D　对　对　错

## 2.4 可变参数

- 在函数定义时，可以设计可变数量参数，通过参数前增加星号（*）实现。
- 调用时，这些参数被当作元组类型传递到函数中，实例如下：

```
>>> def vfunc(a,*b):
        print(type(b))
        for n in b:
            a+= n
        return a
>>> vfunc(1,2,3,4,5)
```

```
>>> def vfunc(a,**b):
        print(type(b))
        print(b)
        return [a,b]
>>> vfunc(1,key_2=3,key_4=5)
```

- 注意：可变参数**只能出现在参数列表的最后**！！

# 2. 函数的参数传递

【思考】程序的运行结果?

```
n=1                          #n是全局变量
def func(a,b):
    c=a*b+n                  #c是局部变量
    return c
s=func(3,5)
print(c)                #局部变量在函数执行完退出时，被释放
```

```
NameError: name 'c' is not defined
```

03

# 变量的作用域

# 3. 变量的作用域

变量作用域的含义：

- 变量**起作用的代码范围**称为变量的作用域，不同作用域内变量名可以相同，互不影响。
- 在函数内部定义的普通变量只在函数内部起作用，称为**局部变量**。当函数执行结束后，局部变量自动删除，不再可以使用。
- 局部变量的引用比全局变量速度快，应优先考虑使用。

# 3. 变量的作用域

局部变量和全局变量

# 3. 变量的作用域

## 局部变量和全局变量

```
n, s = 5, 100
def fact(n) :
    s = 1
    for i in range(1, n+1):
        s *= i
    return s
print(fact(n), s)
```

n和s是全局变量

创建s

fact()函数中的n和s是局部变量

n和s是全局变量

运行结果

```
>>>
120 100
```

## 规则1: 局部变量和全局变量是不同变量

- 局部变量是函数内部的占位符，与全局变量可能重名但不同
- 函数运算结束后，局部变量被释放
- 可以使用*global*保留字在函数内部使用全局变量

# 3. 变量的作用域

```
n, s = 5, 1
def fact(n) :
    global s
    for i in range(1, n+1):
        s *= i
    return s
print(fact(n), s)
```

fact()函数中使用global保留字声明

此处s是全局变量s

此处s指全局变量s

此处全局变量s被函数修改

运行结果

```
>>>
120 120
```

**【举一反三】分析程序运行结果。**

```
n, s = 10, 100
def func(n):
    n=n+1
    return n
print(func(n), n)
```

结果是多少呢?

**11，10**

## 规则2: 局部变量为组合数据类型且未创建，等同于全局变量

```
ls = ["F", "f"]
def func(a) :
    ls.append(a)
    return ls
func("C")
print(ls)
```

通过使用[]真实创建了一个全局变量列表ls

此处ls是列表类型，未真实创建

则等同于全局变量,引用全局变量

全局变量ls被修改

运行结果

```
>>>
['F', 'f', 'C']
```

# 3. 变量的作用域

```
ls = ["F", "f"]
def func(a) :
    ls = []
    ls.append(a)
    return ls
func("C")
print(ls)
```

通过使用[]真实创建了一个全局变量列表ls

此处ls是列表类型，真实创建

ls是局部变量

局部变量ls被修改

运行结果

```
>>>
['F', 'f']
```

## 使用规则

- 基本数据类型，无论是否重名，局部变量与全局变量不同
- 可以通过global保留字在函数内部声明全局变量
- 组合数据类型，如果局部变量未真实创建，则是全局变量

# 3. 变量的作用域

**【对比】分析程序运行结果。**

```
ls = ["F", "f"]
def func(a) :
    ls = []
    ls.append(a)
    return ls
func("C")
print(ls)
```

运行结果

```
>>>
['F', 'f']
```

```
ls = ["F", "f"]
def func(a) :
    ls.append(a)
    return ls
func("C")
print(ls)
```

运行结果

```
>>>
['F', 'f', 'C']
```

```
ls = ["F", "f"]
def func(ls,a) :
    ls.append(a)
    return ls
func(ls,"C")
print(ls)
```

运行结果

```
>>>
['F', 'f', 'C']
```

# 3. 变量的作用域

- 原则：尽量少用全局变量
- 使用全局变量的场景：
  - 多个函数需要修改同一个变量（如计数器）（共享数据）

```
count = 0  # 全局变量
def increment():
    global count  # 声明使用全局变量
    count += 1
def show_count():
    print("当前计数:", count)
increment()
increment()
show_count()  # 输出：当前计数: 2
```

04

# 匿名函数（lambda）

# 4. 匿名函数（lambda）

lambda函数是一种特殊的函数——匿名函数。匿名函数并非没有名字，而是将函数名作为函数结果返回，语法格式如下：

```
<函数名> = lambda <参数列表> : <表达式>
```

lambda函数与正常函数一样，等价于下面形式：

```
def  <函数名>（<参数列表>）:
    return <表达式>
```

# lambda函数

```
>>>f = lambda x,y : x + y      #定义一个lambda函数，执行两个参数相加，结果返回f
>>>print(f(10,20))      #调用上述定义的lambda函数，传入10和20为参数，输出如下

30
```

# 单选题 1分

设置

下列哪个选项是匿名函数的正确用法?

A def lambda x: x*2

B lambda x: x*2

C function(x): return x*2

D lambda(x): x*2

提交

# 4. 匿名函数（lambda）

## ➢匿名函数：用作函数的参数（当参数为函数时）

```
students = [{'name': 'Alice', 'score': 90},
            {'name': 'Bob', 'score': 85}]
new_students=sorted(students, key=lambda x: x['score'])
print(new_students)
```

```
[{'name': 'Bob', 'score': 85}, {'name': 'Alice', 'score': 90}]
```

```
numbers = [1, 2, 3, 4, 5, 6]
# filter()函数返回的是迭代器对象，可以遍历，也可以转换为列表
new_numbers=list(filter(lambda x: x%2==0, numbers))#筛选功能
print(new_numbers)
```

```
[2, 4, 6]
```

# 总结

- 函数定义与调用
- 函数的参数传递
  - 位置传参
  - 关键字传参
  - 默认传参
  - 可变参数
- 变量的作用域
  - 全局变量
  - 局部变量

## 感悟

**人生如函数**：

- ✓ 每个人都在定义自己的人生函数，**参数**是选择，**返回值**是成长。
- ✓ 有些事如**局部变量**只在当下重要，有些如**全局变量**影响一生。
- ✓ 学会为人生函数设置默认值保持初心，用**可变参数**拥抱变化，让每个调用都成为有意义的经历。
- ✓ 编程教会我们的不只是技术，更是**认真生活**的态度。

# 作业

- **使用函数**编写程序打印输出前100个**回文素数**，每行显示10个并对齐，（回文素数指该数**既是素数也是回文数**，如131，313，717），如下所示：

```
2     3     5     7     11    101     131     151     181     191
313   353     373     383     727     757     787     797     919     929
10301     10501     10601     11311     11411     12421     12721     12821     13331     13831
13931     14341     14741     15451     15551     16061     16361     16561     16661     17471
17971     18181     18481     19391     19891     19991     30103     30203     30403     30703
30803     31013     31513     32323     32423     33533     34543     34843     35053     35153
35353     35753     36263     36563     37273     37573     38083     38183     38783     39293
70207     70507     70607     71317     71917     72227     72727     73037     73237     73637
74047     74747     75557     76367     76667     77377     77477     77977     78487     78787
78887     79397     79697     79997     90709     91019     93139     93239     93739     94049
```

# 作业

```
###素数判断
number=int(input())
divisor = 2
while divisor <= number / 2:
    if number % divisor == 0:
        print(number,"非素数")
        break
    divisor += 1
else:
    print(number,"是素数")
```

```
###函数—素数判断
def isPrime(number):
    divisor = 2
    while divisor <= number / 2:
        if number % divisor == 0:
            return False
        divisor += 1
    return True
```

# 作业

```
###回文判断
number=int(input())
result = 0
num=number
while number != 0:
    re = number % 10
    result = result * 10 + re
    number = number // 10
if num == result:
    print(num,'是回文数')
else:
    print(num,'不是回文数')
```

```
###函数-回文判断
def isPal(number):
    result = 0
    num=number
    while number != 0:
        re = number % 10
        result = result * 10 + re
        number = number // 10
    return num == result
```

# 作业

```
###回文素数调用
count = 1
i = 2
while count <= 100:
    if isPrime(i) and isPal(i):
        print("{}\t".format(i), end = " ")
        if count % 10 == 0:
            print()
        count += 1
    i += 1
```

```
2    3    5    7    11   101     131     151     181     191
313  353     373     383     727     757     787     797     919     929
10301    10501    10601    11311    11411    12421    12721    12821    13331    13831
13931    14341    14741    15451    15551    16061    16361    16561    16661    17471
17971    18181    18481    19391    19891    19991    30103    30203    30403    30703
30803    31013    31513    32323    32423    33533    34543    34843    35053    35153
35353    35753    36263    36563    37273    37573    38083    38183    38783    39293
70207    70507    70607    71317    71917    72227    72727    73037    73237    73637
74047    74747    75557    76367    76667    77377    77477    77977    78487    78787
78887    79397    79697    79997    90709    91019    93139    93239    93739    94049
```

# 下课并不代表思考的终止
# 期待我们下次的思想碰撞