Python程序流程控制

1.语句块与缩进

2.if 条件判断 (分支)

3.While/for 循环

C source code

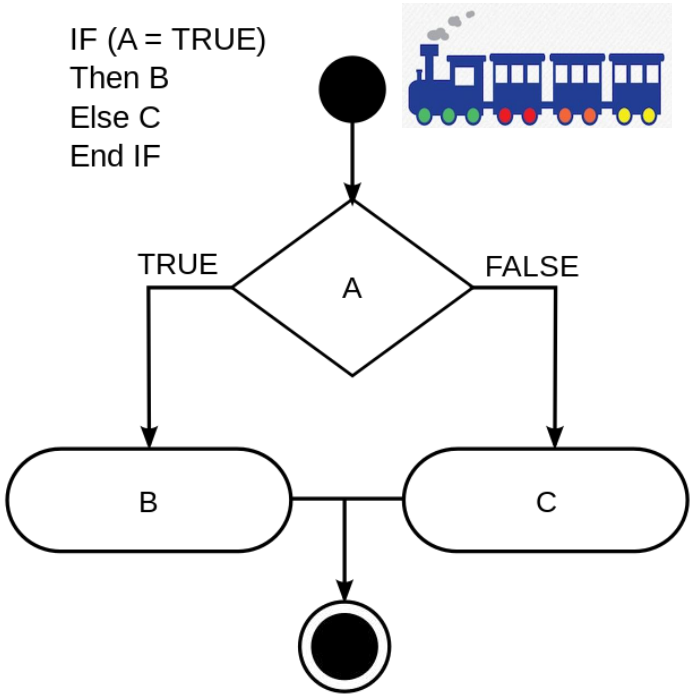
```
#include <stdio.h>

int main () {
    for( ; ; ) {
        printf("Hello!");
    }
    return 0;
}
```

1. 语句块与缩进

```
6  import os
7  import subprocess as sp
8
9  in_ext  = '.mp4'
10 out_ext = '.c.mp4'
11 rootdir = r'E:\录屏\output'
12 os.chdir(rootdir)
13 for parent,dirnames,filenames in os.walk(rootdir):
14     for filename in filenames:
15         current_file = os.path.join(parent,filename)
16         if filename.lower().endswith( in_ext):
17             outfile = current_file[:-4]+out_ext
18             pass
19
20 # do something
```

1 3 4 5 6

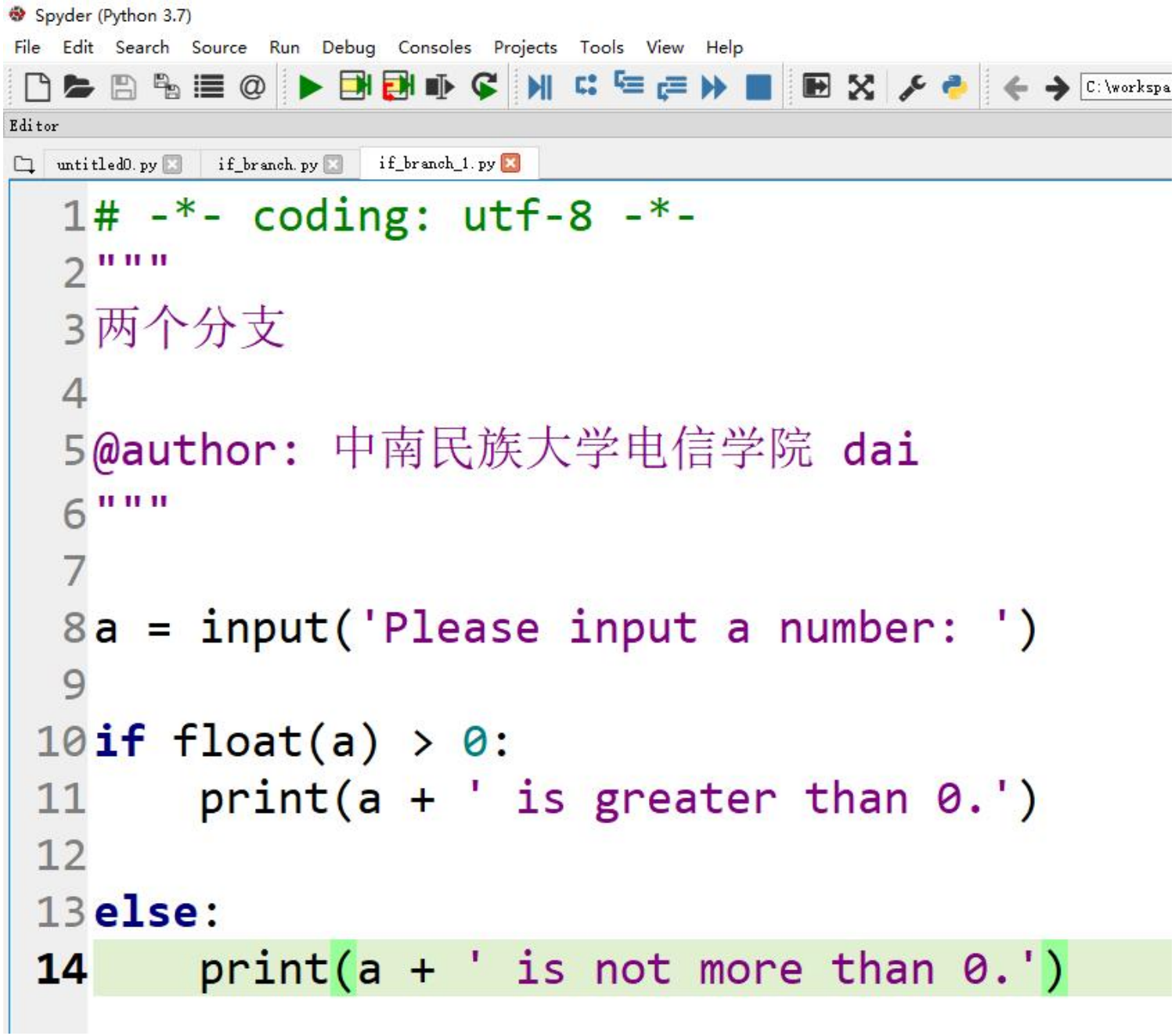
条件判断 if … else …



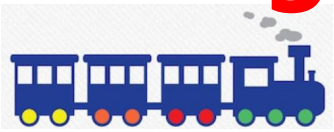

```
if   <condition>:
     …
else:
     …
```

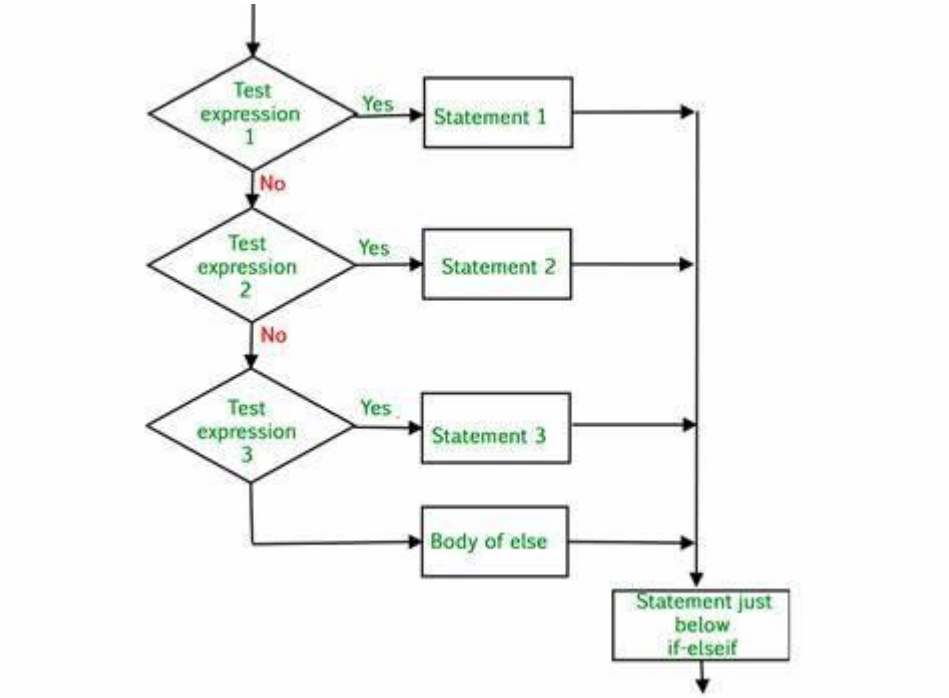
【例】判断(一个数a是否大于零):

- 条件为真, 输出: 1
- 否则输出: -1


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\workspa
Editor
untitled0.py
if_branch.py
if_branch_1.py
1# -*- coding: utf-8 -*-
2"""
3两个分支
4
5@author: 中南民族大学电信学院 dai
6"""
7
8a = input('Please input a number: ')
9
10if float(a) > 0:
11   print(a + ' is greater than 0.')
12
13else:
14   print(a + ' is not more than 0.')

3个或更多分支 if … elif … else…

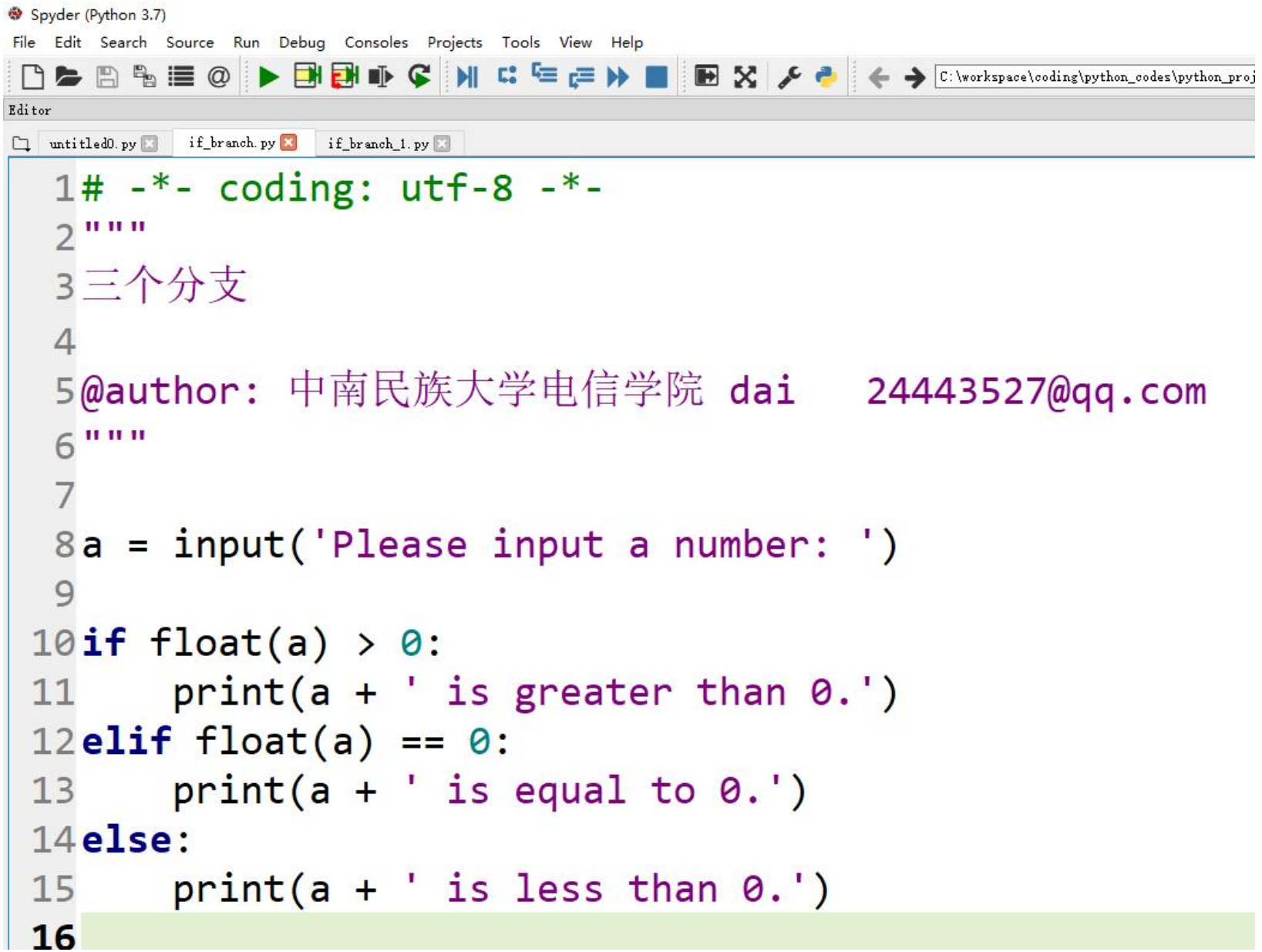





```
if  <condition>:
      …
elif:
      …
elif:
      …
else:
      …
```

【例】判断1个数a是否大于零：

- 如果 a >0, 输出: 正数
- 否则如果 a == 0,输出 : 0
- 否则, 输出: 负数

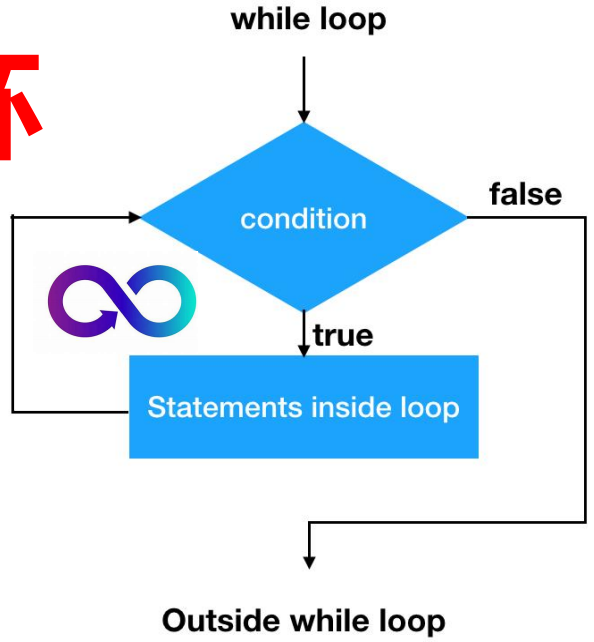

Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\workspace\coding\python_codes\python_proj
Editor
untitled0.py
if_branch.py
if_branch_1.py
1 # -*- coding: utf-8 -*-
2 """
3 三个分支
4
5 @author: 中南民族大学电信学院 dai 24443527@qq.com
6 """
7
8 a = input('Please input a number: ')
9
10 if float(a) > 0:
11 print(a + ' is greater than 0.')
12 elif float(a) == 0:
13 print(a + ' is equal to 0.')
14 else:
15 print(a + ' is less than 0.')
16

Switch case 怎么弄?

```
switch case:
    …
case 1:
    …
case 2:
    …
case 3:
    …
```

```
if case==1 :
    …
elif case==2 :
    …
elif case==3 :
    …
```

While 循环



- 打印 1 到 100 的所有整数

```
x = 1
while x <= 100:
    print(x)
    x += 1
```

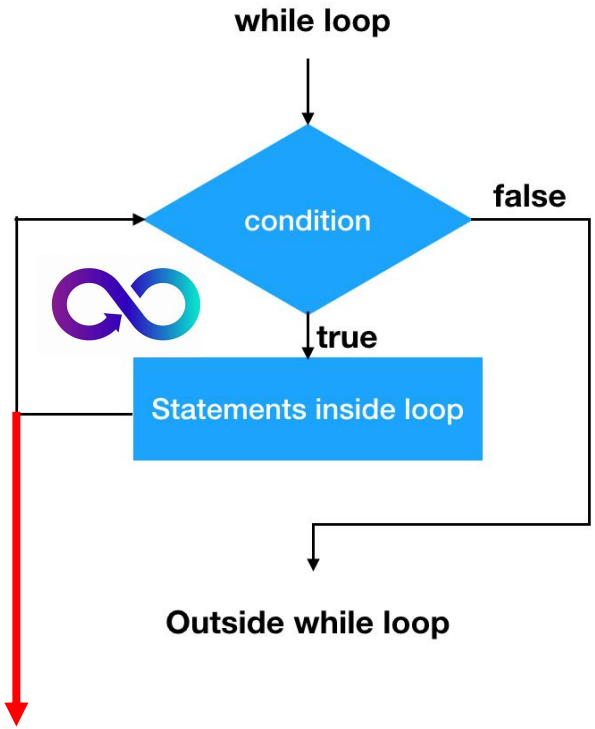
- 【例】提示用户输入用户名，然后打印”Hello, xxx” 的欢迎消息。需要注意的是，如果用户不给任何输入，直接回车时，要继续提示用户输入。

```
name = ''
while not name:
    name = input('Please enter your name: ')
print('Hello, {}!'.format(name))
```

Break 主动中止循环

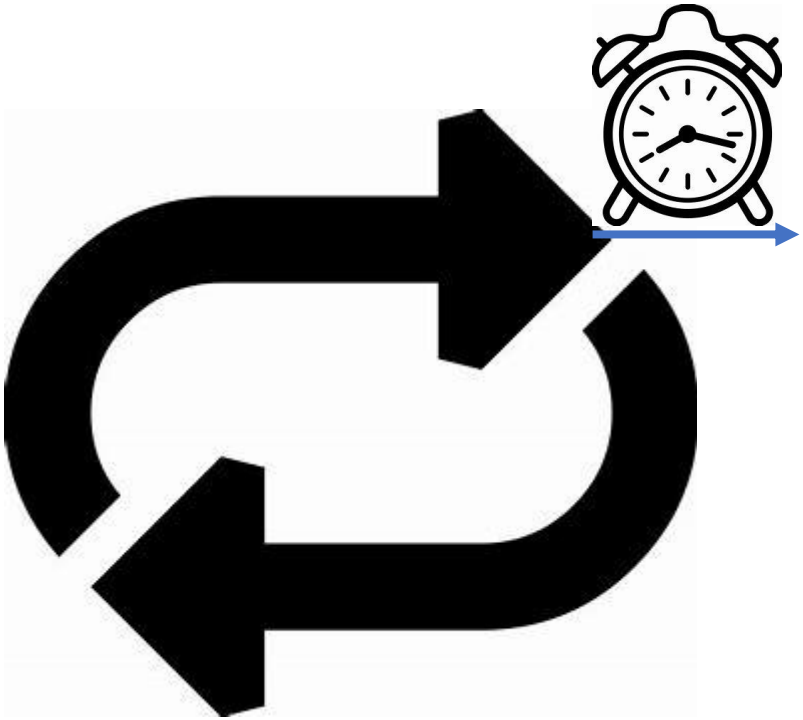
- 【例】打印 1 到 100 的所有整数 的另一种写法

```
x = 1
while True:
    if x > 100:
        break

    print(x)
    x += 1

# 此处循环已结束
```



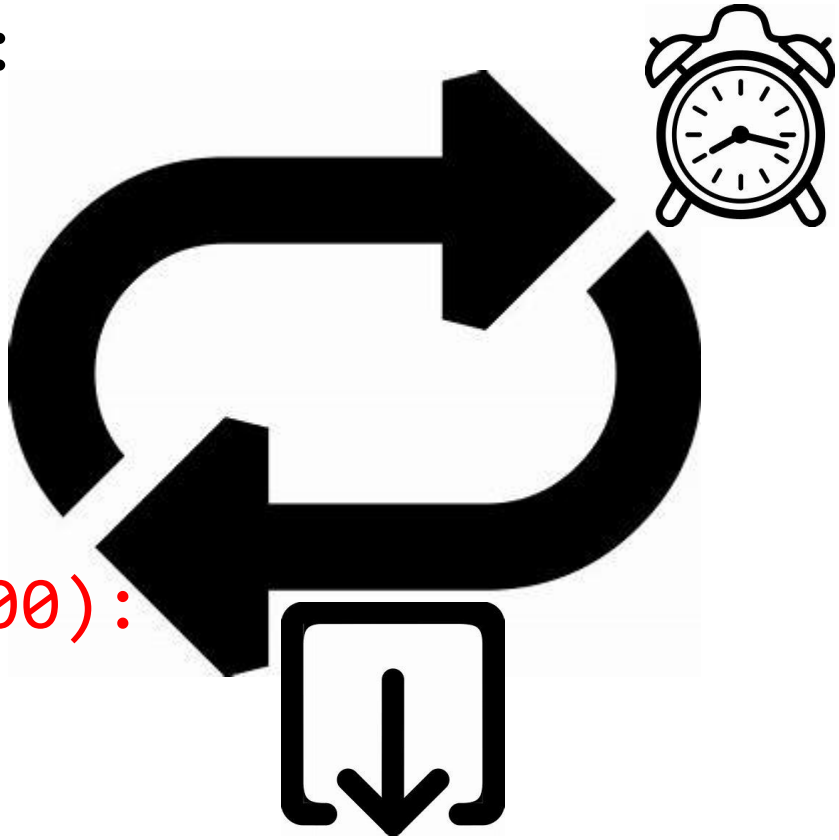
For … in … 循环, range()

1. 对全班中的每位同学执行xxx
2. 对数据库中的每条执行xxx
3. 对文件的每一行执行xxx
4. 对 excel 中的每一行执行xxx
5. 对于文本中的每一句xxx
6. 对列表[1,2,3,…10]中的每个元素，去执行xxx



【例】打印0-100的整数的 for 循环版本

```
for each in range(1,101):
    print(each)
```



```
for each in range(1,100000):
    print(each)
    if each > 100:
        break
```

打印 1-10 整数的平方数

```
8 for n in range(10):
9     print(n, n**2)
```

实例: XML格式化文本

- 已有一个已初步格式化的 txt 文档，它由汉语土家语对照的句子组成。现在希望能够用一套 XML 标签对其进行标记，方便其它软件使用。

```
pa55 je55 phan21 toŋ35 zi35 tiu55
吧咂盘冬做来 ，

pi35 pi35 tʰhi55 pa51 en35 tʰiu35
小的大的来在，

lo35 pu55 tʰhu55 tʰhu21 pa51 po21 la21
眼睛睁睁看着在，

...


翻译：
吧咂盘冬，莫做了，
嘀嘀咂嘀，莫唱了，
过了正月元宵节，
跳跳唱唱要停歇。
十五吃了爬坡肉(注)，
...
```

```
<passage>
  <title>
    <zh>打谷子 </zh>
  </title>
  <s id="1">
    <tu>ten21 toŋ21 ten21 toŋ21 oŋ21 la21 xu55</tu>
    <zh> 叮冬叮冬响在了。 </zh>
  </s>
  <s id="2">
    <tu>ten21 toŋ21 ten21 toŋ21 xa21 la21 xu55</tu>
    <zh> 叮冬叮冬响在了， </zh>
  </s>
  <s id="3">
    <tu>li35 pu55 xa21 ma21 xa21 la21 xu55</tu>
    <zh> 谷子打人打在了。 </zh>
  </s>
  <s id="4">
    <tu>ŋa51 ma51 le55</tu>
    <zh> 割的人哩， </zh>
  </s>
  <s id="5">
    <tu>se21 la21 me35 ka21 tʰhiau35 po55 la55</tu>
    <zh> 屁股天上翘着在， </zh>
  </s>
  <s id="6">
    <tu>kho55 pa51 pa21 tsi21 ka35 po51 la55</tu>
    <zh> 脑壳泥巴吃着在。 </zh>
  </s>
  <trans>
    <p id="1">阵阵秋风阵阵凉，</p>
    <p id="2">田里谷子一片黄，</p>
    <p id="3">叮冬叮冬扮桶响，</p>
    <p id="4">毕兹卡收割忙又忙。</p>
  </trans>
</passage>
```

简单补充主要知识点

- 打开文件,读行，逐行读;
- 字符串文本的分割方法：按特殊字符分，空行，换行
- Xml 的基本使用方法：BeautifulSoup
- 字符串模板

定义函数

作用：为了模块化和复用 代码

1.带参数 / 不带参数

2.位置参数/ 名字匹配/ 参数值缺省

3.有返回值 / 无返回值

打印 10 遍 “hello”

```
def print_hello():
    print(“hello”*10)
```

打印10次

```
def print_hello(n):
    print(“hello”*n)
```

打印n次

```
def print_hello(n=10):
    print(“hello”*n)
```

打印n次，默认10次

```
def print_any(s, n=10):
    print(s*n)
```

打印内容可变

返回一些状态信息

```
def print_any(s, n=10):
    length = len(s)*n
    print(s*n)
    return length
```

返回打印的字符串长度

```
def print_any(s, n=10):
    result = s * n
    print(s*n)
    return length, s*n
```

返回更多的信息

```
n,s = print_any(“hello”, n=10)
```

增加参数不影响已有的函数调用

```
def print_repeat(s, n=10):
    print(s*n)
```

```
hellohellohello
```

```
hello-hello-hello-
```

例：增加一个连接符选项

```
def print_repeat(s, n=10, sep = “”):
    print((s+sep)*n)
```

增加返回值的问题

```
def print_any(s, n=10):
    result = s * n
    return length
```

```
def print_any(s, n=10):
    result = s * n
    return length, s*n
```

演示: 增加函数返回值后，会造成所有的结果 unpack 都不再正确，需要手工改写代码

函数返回多个值

```
def multi_returns():
    pass
    return res1, res2
```

接收返回值方法1

```
C, D = multi_returns()
```

接收返回值方法2

```
A = multi_returns()
C = A[0]
D = A[1]
```

下划线变量名接收返回值

旧函数

```
def print_any(s, n=10):
    result = s * n
    return length
```

新函数

```
def print_any(s, n=10):
    result = s * n
    return length, s*n
```

旧调用

```
C= print_any()
```

新调用

```
C,D = print_any()
```

新调用

```
X = print_any()
C = X[0]
D = X[1]
```

增加函数返回值后，会造成所有的结果 unpack 都不再正确，需要手工改写代码

举例：用 dict 返回参数

```
def print_any(s, n=10):
    result = s * n
    length=len(result)
    return length
```

升级后影响已有调用

```
def print_any(s, n=10):
    result = s * n
    length=len(result)
    return length, s*n
```

用 dict 包装返回值

```
def print_any(s, n=10):
    result = s * n
    length=len(result)
    res={“result”: result;
        “length”: length}
    return res
```

返回 dict 的简化版

```
def print_any(s, n=10):
  return{
    “result”: s * n;
    “length”:len(result)}
```

例: AWGN 函数编写

利用正态分布的随机数，产生一个AWGN。它可以：

- 可不带任何参数，此时它返回一个正态分布的单个取样值。
- 还可以指定它的**长度**，不指定时长度为1
- 可指定**均值**,当不指定噪声的均值时，其均值为0
- 可指定噪声的**方差**, 当方差不指定时，方差为1

字典、json

字典的迭代

基于 Json 的数据服务API

Mysql 数据服务

什么是Python的模块 module

- 什么是模块?
- 模块有何特点?
- 导入模块的方法
- 自定义模块的方法

引入模块

- ✓ **import** module_name1, module_name1,
- ✓ **import** module_name **as** mn
- ✓ **from** module **import** func1, func2, func3
- x **from** module **import** *

自定义的模块

1. 自定义的模块以及调用的方法
2. 怎样将测试代码写在模块中
3. 怎样才能保证可以成功导入自定义模块，

```
# mymodule.py
def print_hello():
    print('Hello, nice to meet you!')
```

```
import mymodule

mymodule.print_hello()
```

引入模块

- ✓ **import** module_name1, module_name1
- ✓ **import** module_name **as** mn
- ✓ **from** module **import** func1, func2, func3
- x **from** module **import** *

怎样将测试代码写在模块中

- 运行本文件时，测试代码会运行
- 从其它文件导入本模块文件时，只导入函数，测试代码不运行

```
def print_hello():
    print('Hello, nice to meet you!')
print_hello()
```

```
def print_hello():
    print('Hello, nice to meet you!')
if __name__ == '__main__':
    print_hello()
```

怎么把写好的模块共享给别人用?

办法: 将 module 文件放在 Python 的搜索路径

1. 当前目录
2. 如果不在当前目录，Python 则搜索在 shell 变量 PYTHONPATH 下的每个目录。
3. 如果都找不到，Python会察看默认路径。UNIX下，默认路径一般为/usr/local/lib/python/。

作业: 同名模块的问题

- 假设你针对某个函数写了多个同名模块，它们的内容因此并不完全相同。那么你在 import 它们时，要注意什么问题, 怎么解决这些问题?