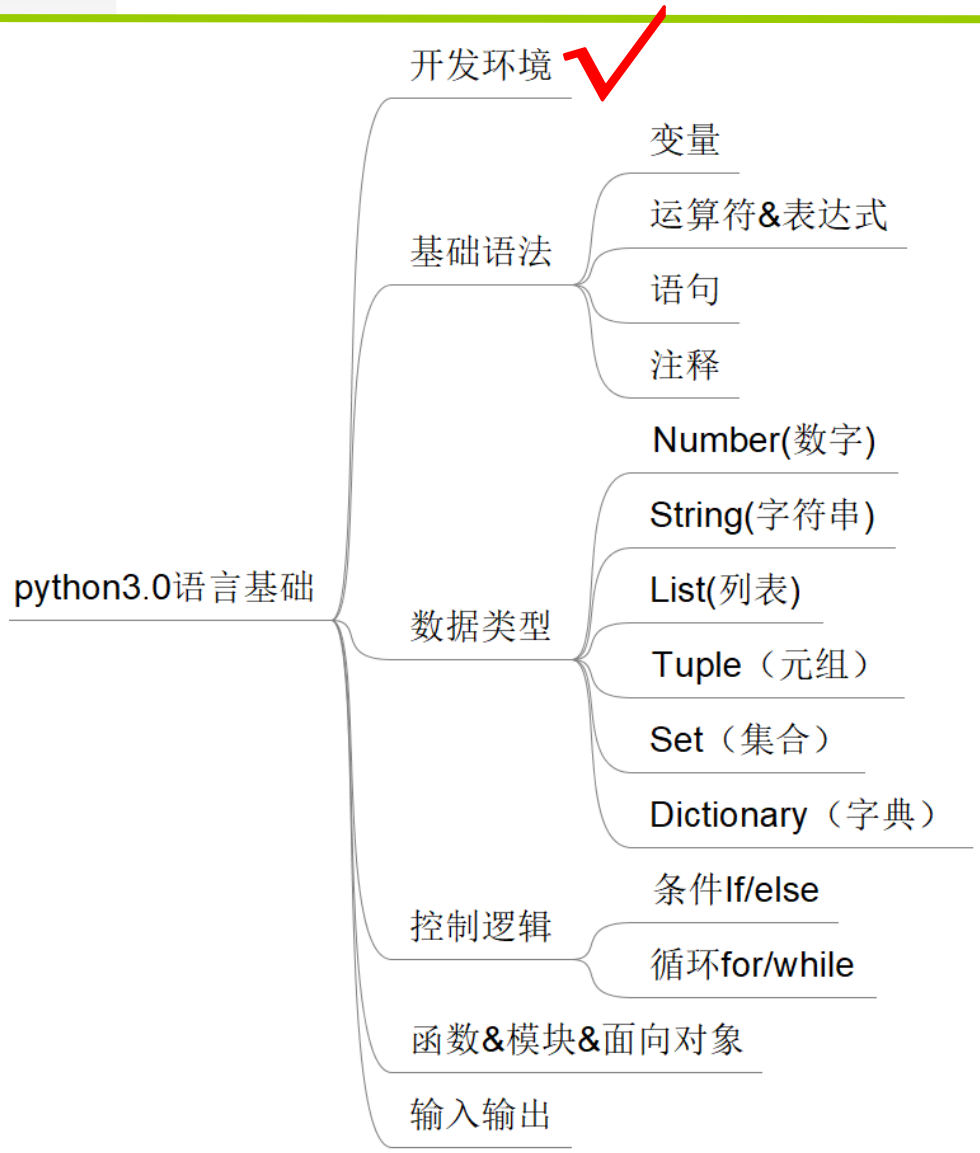
Python语言

02 Python3.0语言基础

广东工业大学自动化学院　邢延

内容提要

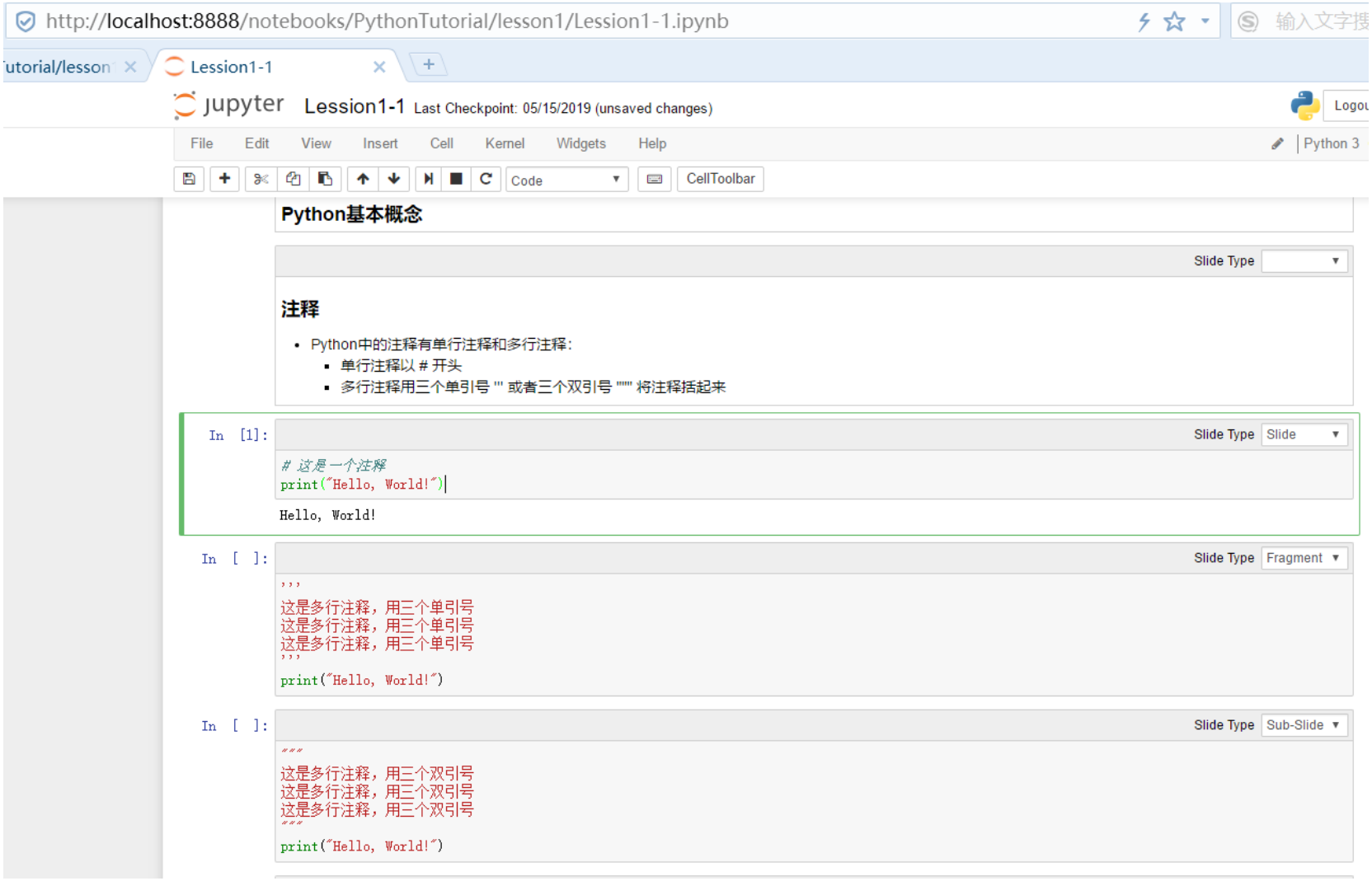

python3.0语言基础
开发环境
基础语法
变量
运算符&表达式
语句
注释
数据类型
Number(数字)
String(字符串)
List(列表)
Tuple（元组）
Set（集合）
Dictionary（字典）
控制逻辑
条件If/else
循环for/while
函数&模块&面向对象
输入输出

1. 开发环境

- JupyterNotebook
  - Web页面，交互式，文档代码混合模式
- PyCharm IDE
  - 类似传统Eclipse开发工具，适合开发复杂的Python项目，包含多个文件，多个模块

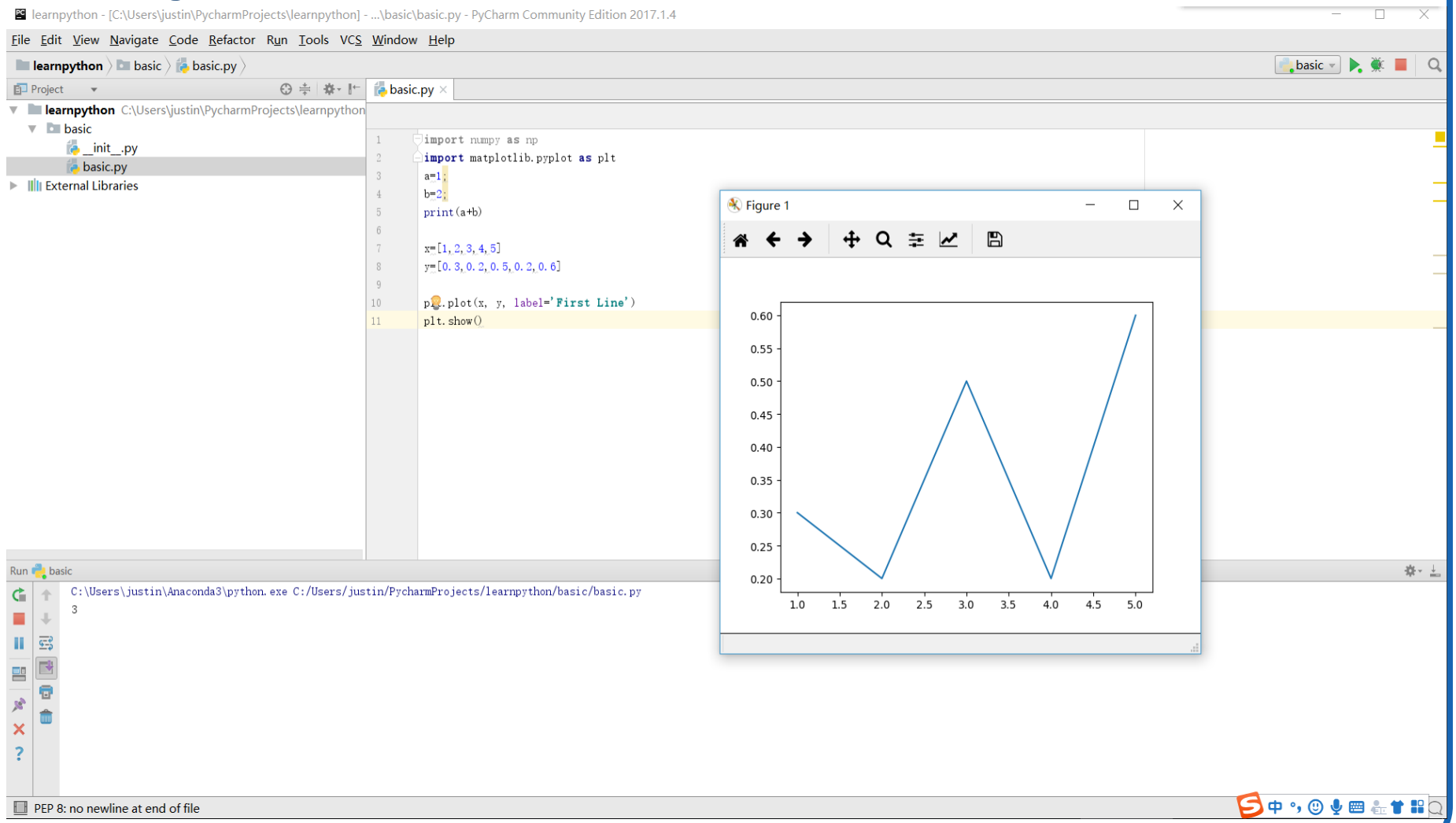
1. 开发环境

JupyterNotebook

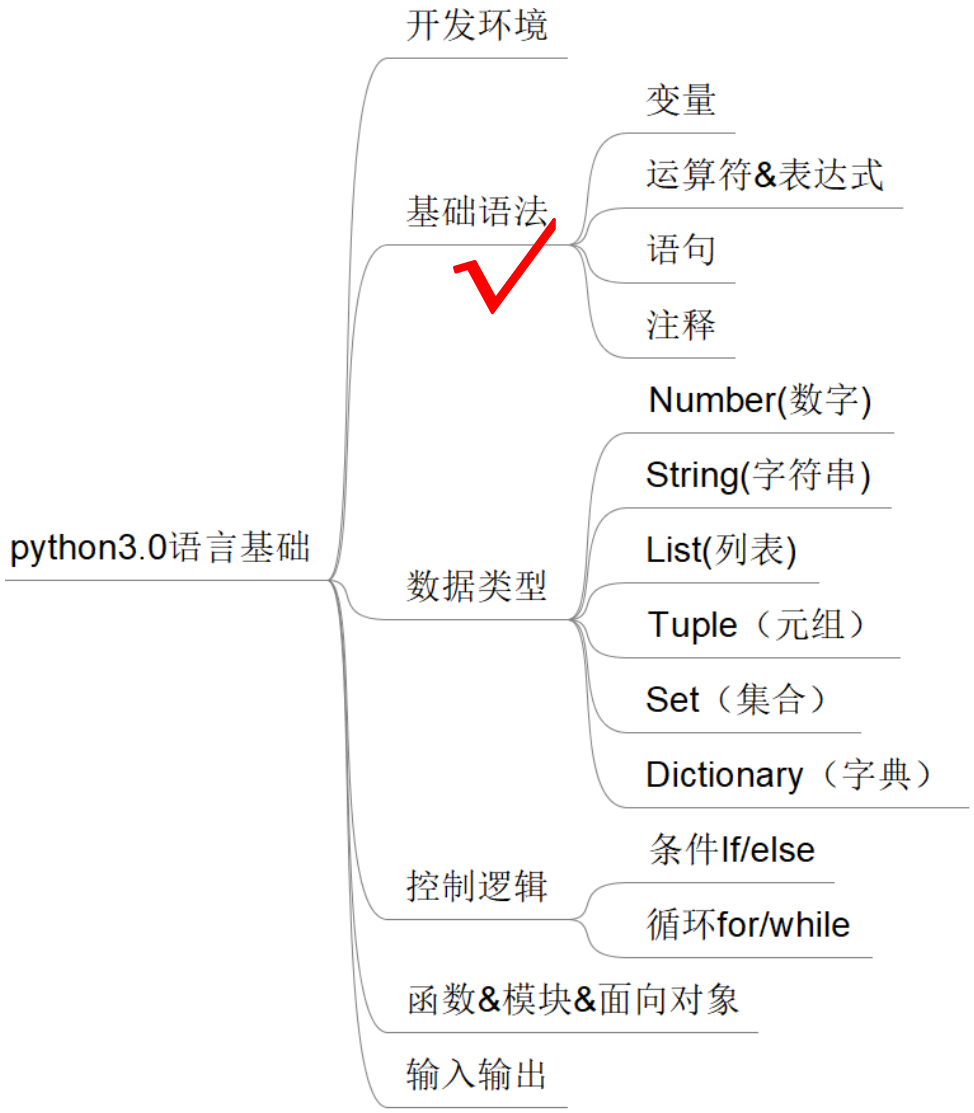


1. 开发环境

❖ Pycharm



内容提要


python3.0语言基础
开发环境
基础语法
变量
运算符&表达式
语句
注释
数据类型
Number(数字)
String(字符串)
List(列表)
Tuple（元组）
Set（集合）
Dictionary（字典）
控制逻辑
条件If/else
循环for/while
函数&模块&面向对象
输入输出

2. 基础语法

实例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 例1：if 基本用法

flag = False
name = 'luren'
if name == 'python':          # 判断变量是否为 python
    flag = True               # 条件成立时设置标志为真
    print 'welcome boss'      # 并输出欢迎信息
else:
    print name                # 条件不成立时输出变量名称
```

每一行为一条语句

赋值

变量

注释

条件判断

缩进

打印输出到命令行

输出结果为:

```
luren           # 输出结果
```

2. 基础语法

❖ 变量与变量名（标识符）

python保留字

保留字即关键字，我们不能把它们用作任何标识符名称。Python 的标准库提供了一个 keyword 模块，可以输出当前版本的所有关键字：

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'f
inally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'retur
n', 'try', 'while', 'with', 'yield']
```

2. 基础语法

❖ 注释

Python中单行注释以 # 开头，实例如下：

实例(Python 3.0+)

```
#!/usr/bin/python3

# 第一个注释
print ("Hello, Python!") # 第二个注释
```

执行以上代码，输出结果为：

```
Hello, Python!
```

多行注释可以用多个 # 号，还有 ''' 和 """：

实例(Python 3.0+)

```
#!/usr/bin/python3

# 第一个注释
# 第二个注释

'''
第三注释
第四注释
'''

"""
第五注释
第六注释
"""
print ("Hello, Python!")
```

执行以上代码，输出结果为：

```
Hello, Python!
```

2. 基础语法

❖ 缩进

实例(Python 3.0+)

```
if True:
    print ("True")
else:
    print ("False")
```

以下代码最后一行语句缩进数的空格数不一致，会导致运行错误：

```
if True:
    print ("Answer")
    print ("True")
else:
    print ("Answer")
  print ("False")    # 缩进不一致，会导致运行错误
```

2. 基础语法

❖ 多行语句

语句是Python解释器可以运行的一个代码单元，也可以理解为可以执行的命令

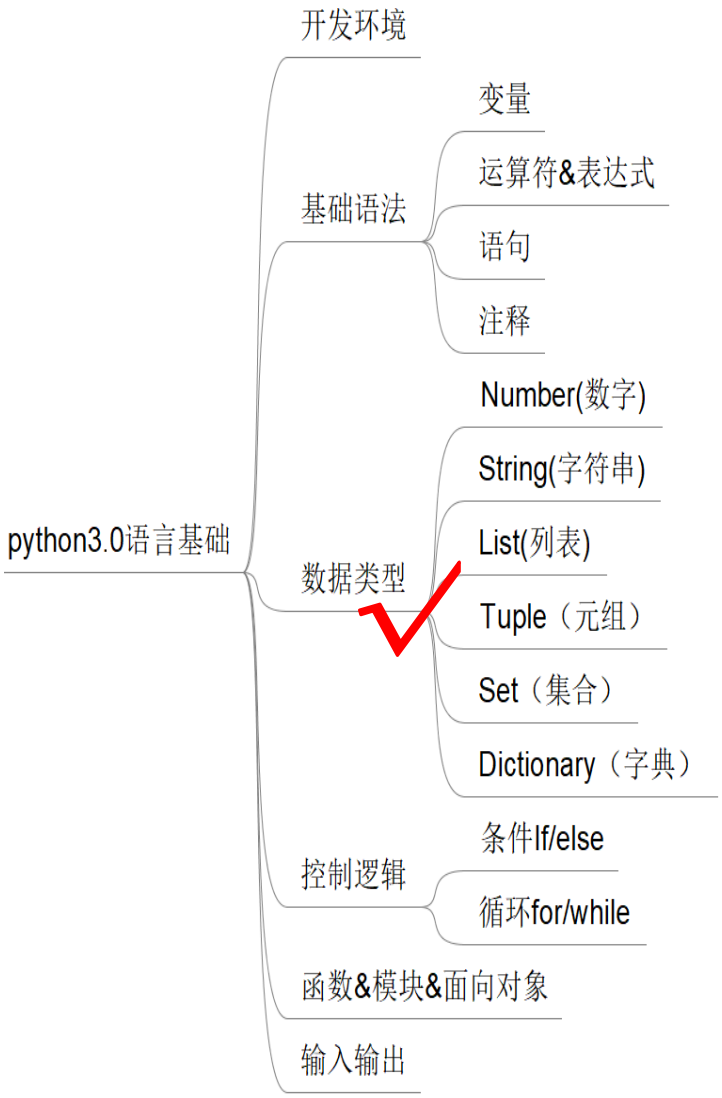
Python 通常是一行写完一条语句，但如果语句很长，我们可以使用反斜杠(\)来实现多行语句，例如：

```
total = item_one + \
        item_two + \
        item_three
```

在 [], {}, 或 () 中的多行语句，不需要使用反斜杠(\)，例如：

```
total = ['item_one', 'item_two', 'item_three',
        'item_four', 'item_five']
```

内容提纲


python3.0语言基础
开发环境
基础语法
变量
运算符&表达式
语句
注释
数据类型
Number(数字)
String(字符串)
List(列表)
Tuple（元组）
Set（集合）
Dictionary（字典）
控制逻辑
条件If/else
循环for/while
函数&模块&面向对象
输入输出

3. 数据类型

- Python3 中有六个标准的数据类型
  - 基础类型：
    - Number（数字）
    - String（字符串）
  - 数据结构：
    - List（列表）
    - Tuple（元组）
    - Set（集合）
    - Dictionary（字典）

3. 数据类型

- 数字
  - 整型(Int)
    - Python3 没有限制整型大小，可以当作 Long 类型使用。
  - 浮点型(float)
    - 由整数部分与小数部分组成，也可以使用科学计数法表示（2.5e2 = 2.5 x 10^2 = 250）
  - 复数((complex))
    - 由实数部分和虚数部分构成，可以用a + bj， 或者complex(a,b)表示；
    - 复数的实部a和虚部b都是浮点型。

3. 数据类型

❖ 算数运算符

假设变量a为10，变量b为21

算术运算符	描述	实例
+	加 - 两个对象相加	a + b 输出结果 31
-	减 - 得到负数或是一个数减去另一个数	a - b 输出结果 -11
*	乘 - 两个数相乘或是返回一个被重复若干次的字符串	a * b 输出结果 210
/	除 - x 除以 y	b / a 输出结果 2.1
%	取模 - 返回除法的余数	b % a 输出结果 1
**	幂 - 返回x的y次幂	a**b 为10的21次方

//	取整除 - 向下取接近除数的整数	>>> 9//2
		4
		>>> -9//2
		-5

3. 数据类型

❖ 赋值运算符

假设变量a=10，变量b=20

赋值运算符	描述	实例
=	简单的赋值运算符	c = a + b 将 a + b 的运算结果赋值为 c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c **= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c // a

3. 数据类型

❖ 逻辑运算符

假设变量a=10，变量b=20

运算符	逻辑表达式	描述	实例
And	x and y	布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。	(a and b) 返回 20。
Or	x or y	布尔“或” - 如果 x 是非 0，它返回 x 的值，否则它返回 y 的计算值。	(a or b) 返回 10。
Not	not x	布尔“非” - 如果 x 为 True，返回 False 。如果 x 为 False，它返回 True。	not(a and b)返回False

3. 数据类型

- 复数((complex))
  - 由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示，复数的实部a和虚部b都是浮点型。

```
a=4.7+0.666j                #定义一个虚数
print(a)                    #输出这个虚数
print(a.real)               #输出实部
print(a.imag)               #输出虚部
print(a.conjugate())        #输出该复数的共轭复数
```

3. 数据类型

数字类型转换

- **int(x)** 将x转换为一个整数。
- **float(x)** 将x转换到一个浮点数。
- **complex(x)** 将x转换到一个复数，实数部分为 x，虚数部分为 0。
- **complex(x, y)** 将 x 和 y 转换到一个复数，实数部分为 x，虚数部分为 y。x 和 y 是数字表达式。

int(x)示例：

```
>>> int(352.1)
352
```

float(x)示例：

```
>>> float(352.1)
352.1
>>> float(int(352.1))
352.0
```

3. 数据类型

❖ 字符串

是 Python 中最常用的数据类型，用引号(' 或 ")来创建字符串。

创建字符串很简单，只要为变量分配一个值即可。

Python 访问子字符串，可以使用方括号来截取字符串。

实例(Python 3.0+)

```
#!/usr/bin/python3

var1 = 'Hello World!'
var2 = "Runoob"

print ("var1[0]: ", var1[0])
print ("var2[1:5]: ", var2[1:5])
```

以上实例执行结果:

```
var1[0]:  H
var2[1:5]:  unoo
```

3. 数据类型

❖ 字符串

下表实例变量a值为字符串 "Hello"，b变量值为 "Python"：

操作符	描述	实例
+	字符串连接	a + b 输出结果： HelloPython
*	重复输出字符串	a*2 输出结果：HelloHello
[]	通过索引获取字符串中字符	a[1] 输出结果 **e**
[:]	截取字符串中的一部分，遵循**左闭右开**原则，str[0,2] 是不包含第 3 个字符的。	a[1:4] 输出结果 **ell**
in	成员运算符 - 如果字符串中包含给定的字符返回 True	**'H' in a** 输出结果 True
not in	成员运算符 - 如果字符串中不包含给定的字符返回 True	**'M' not in a** 输出结果 True
r/R	原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。 原始字符串除在字符串的第一个引号前加上字母 `r`（可以大小写）以外，与普通字符串有着几乎完全相同的语法。	`print( r'\n' )` `print( R'\n' )`
%	格式字符串	请看下一节内容。

3. 数据类型

❖ 字符串

在需要在字符中使用特殊字符时，python用反斜杠(\)转义字符。如右表。

字符串的单引号和双引号都无法取消特殊字符的含义，如果想让引号内所有字符均取消特殊意义，在引号前面加r，如name = r'l\thf'

转义字符	描述
\(在行尾时)	续行符
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，yy代表的字符，例如：\o12代表换行
\xyy	十六进制数，yy代表的字符，例如：\x0a代表换行
\other	其它的字符以普通格式输出

3. 数据类型

❖ 字符串

格式化字符串：将数值或字符串等按照格式插入到字符串指定位置中

实例(Python 3.0+)

```
#!/usr/bin/python3

print ("我叫 %s 今年 %d 岁!" % ('小明', 10))
```

以上实例输出结果:

```
我叫 小明 今年 10 岁!
```

符　号	描述
%c	格式化字符及其ASCII码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化浮点数字，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	%f和%e的简写
%G	%f 和 %E 的简写
%p	用十六进制数格式化变量的地址

3. 数据类型

❖ 字符串

字符串函数（部分）

大小写处理

函数	作用	示例	输出
capitalize	首字母大写，其余小写	'lk with psr'.capitalize()	'Lk with psr'
upper	全部大写	'lk with psr'.upper()	'LK WITH PSR'
lower	全部小写	'lk with psr'.lower()	'lk with psr'
swapcase()	大小写互换	'Lk with Psr'.swapcase()	'lK WITH pSR'
.title()	首字母大写	'lk with psr'.title()	'Lk With Psr'

字符串替换

函数	作用	示例	输出
replace('old','new')	替换old为new	'hello world'.replace('world','python')	hello python
replace('old','new',次数)	替换指定次数的old为new	'hello world'.replace('l','p',2)	heppo world

3. 数据类型

❖ 字符串

字符串函数（部分）

用字符串中的特定符分割字符串

函数	作用	示例	输出
split()	默认按空格分隔	' h e-l lo ' .split()	['h', 'e-l', 'lo']
split('指定字符')	按指定字符分割字符串为数组	' h e-l lo ' .split('-')	[' h e', 'l lo ']

搜索

函数	作用	示例	输出
find()	搜索指定字符串，没有返回-1	'lk la'.find('lk')	0
index()	同上，但是找不到会报错	'lk la'.index('lk')	0
rfind()	从右边开始查找	'lk la'.rfind('lk')	0
count()	统计指定的字符串出现的次数	'lklklk'.count('lk')	3

3. 数据类型

❖ 数据结构 – 序列

- 序列是Python中最基本的数据结构。序列中的每个元素都分配一个数字 - 它的位置，或索引，第一个索引是0，第二个索引是1，依此类推

	举例	特点
List	list1=[1,2,3,4]; list2=['apple','huawei',4]	列表的数据项不需要具有相同的类型
Tuple	Tup1=(1,2,3,4); Tup2=('apple','huawei','xiaomi'); tup3 = "a", "b", "c", "d";	元组与列表类似，不同之处在于元组的元素初始化后不能修改
Set	>>>basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'} >>> print(basket) # 这里演示的是去重功能 {'orange', 'banana', 'pear', 'apple'}	集合（set）是一个无序的不重复元素序列
Dictionary	dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}	键必须是唯一的，但值则不必。 值可以取任何数据类型，但键必须是不可变的，如字符串，数字或元组。

3. 数据类型

❖ 数据结构 – 列表List

访问列表中的值

使用下标索引来访问列表中的值，同样你也可以使用方括号的形式截取字符，如下所示：

实例(Python 3.0+)

```
#!/usr/bin/python3

list1 = ['Google', 'Runoob', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];

print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

运行实例 »

以上实例输出结果：

```
list1[0]:  Google
list2[1:5]:  [2, 3, 4, 5]
```

更新列表

你可以对列表的数据项进行修改或更新，你也可以使用append()方法来添加列表项，如下所示：

实例(Python 3.0+)

```
#!/usr/bin/python3

list = ['Google', 'Runoob', 1997, 2000]

print ("第三个元素为 : ", list[2])
list[2] = 2001
print ("更新后的第三个元素为 : ", list[2])
```

注意：我们会在接下来的章节讨论append()方法的使用

以上实例输出结果：

```
第三个元素为 :  1997
更新后的第三个元素为 :  2001
```

3. 数据类型

❖ 数据结构 – 列表List

Python列表脚本操作符

列表对 + 和 * 的操作符与字符串相似。+ 号用于组合列表，* 号用于重复列表。

如下所示:

Python 表达式	结果	描述
len([1, 2, 3])	3	长度
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	组合
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	重复
3 in [1, 2, 3]	True	元素是否存在于列表中
for x in [1, 2, 3]: print(x, end=" ")	1 2 3	迭代

3. 数据类型

- 数据结构 – 列表List

Python函数

序号	函数	示例
1	len(list) 列表元素个数	>>>list1 = ['a', 'b', 'c'] >>>Len(list1) 3
2	max(list) 返回列表元素最大值	
3	min(list) 返回列表元素最小值	
4	list(seq) 将元组转换为列表	

3. 数据类型

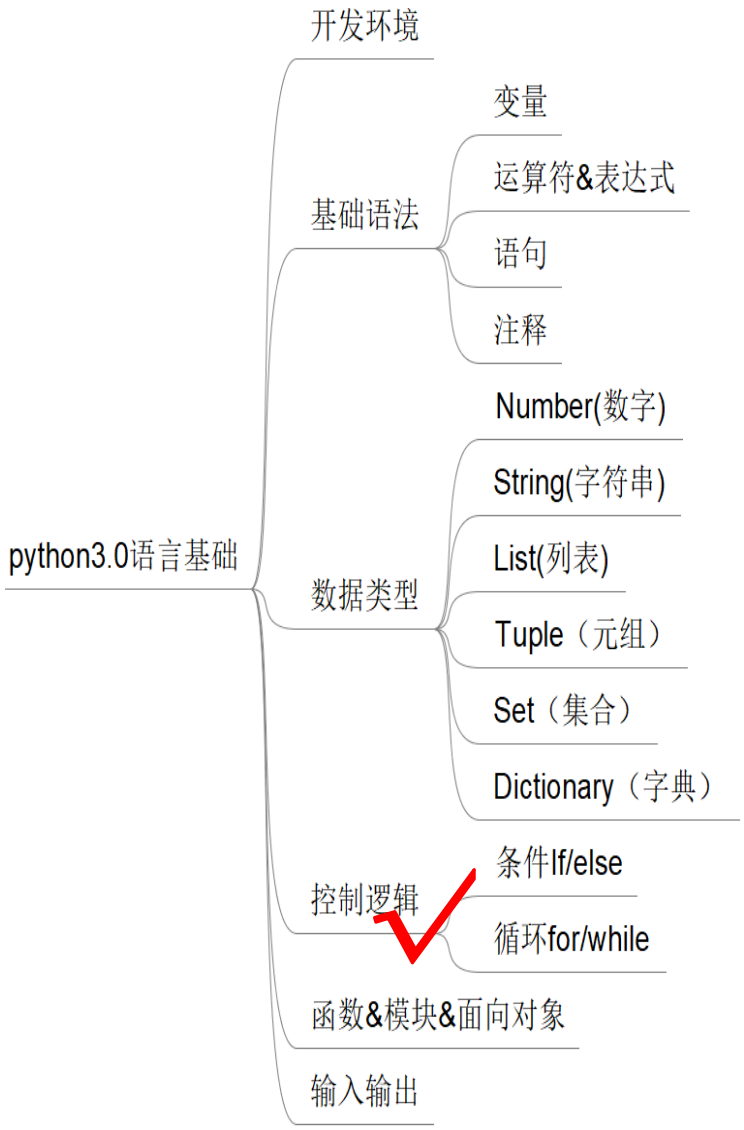
❖ 数据结构 – 列表List

序号	方法（list类型的方法）		
1	list.append(obj)	在列表末尾添加新的对象	>>>list1 = ['a', 'b', 'c'] >>>list1.append(3) ['a', 'b', 'c',3]
2	list.count(obj)	统计某个元素在列表中出现的次数	
3	list.extend(seq)	在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）	
4	list.index(obj)	从列表中找出某个值第一个匹配项的索引位置	
5	list.insert(index, obj)	将对象插入列表	
6	list.pop([index=-1])	移除列表中的一个元素（默认最后一个元素），并且返回该元素的值	
7	list.remove(obj)	移除列表中某个值的第一个匹配项	
8	list.reverse()	反向列表中元素	
9	list.sort(key=None, reverse=False)	对原列表进行排序	
10	list.clear()	清空列表	
11	list.copy()	复制列表	

3. 数据类型

- 其他几种序列的操作有一些和list是类似的，也有些自己特有的函数，请使用时参考官方教程

内容提纲


python3.0语言基础
开发环境
基础语法
变量
运算符&表达式
语句
注释
数据类型
Number(数字)
String(字符串)
List(列表)
Tuple（元组）
Set（集合）
Dictionary（字典）
控制逻辑
条件If/else
循环for/while
函数&模块&面向对象
输入输出

4. 控制语句

❖ 语句块

- 语句块是在满足一定条件下执行一次或多次的一组语句。
- 语句块的创建方式为在代码前放置空格缩进。
- 同一段语句块中的每行都要保持同样的缩进，若没有，Python编译器回认为不属于同一语句块或是认为错误。
- 在Python中，冒号（:）用来标识语句块的开始，块中的每一个语句都是缩进的（缩进量相同）。当退回到和已经闭合的块一样的缩进量时，就表示当前块已经结束了。

```
var1 = 100
if var1:
            print ("1 - if 表达式条件为 true")
            print (var1)
var2 = 0
```

4. 控制语句

❖ 条件控制

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

•1、每个条件后面要使用冒号 :，表示接下来是满足条件后要执行的语句块。
•2、使用缩进来划分语句块，相同缩进数的语句在一起组成一个语句块。

实例

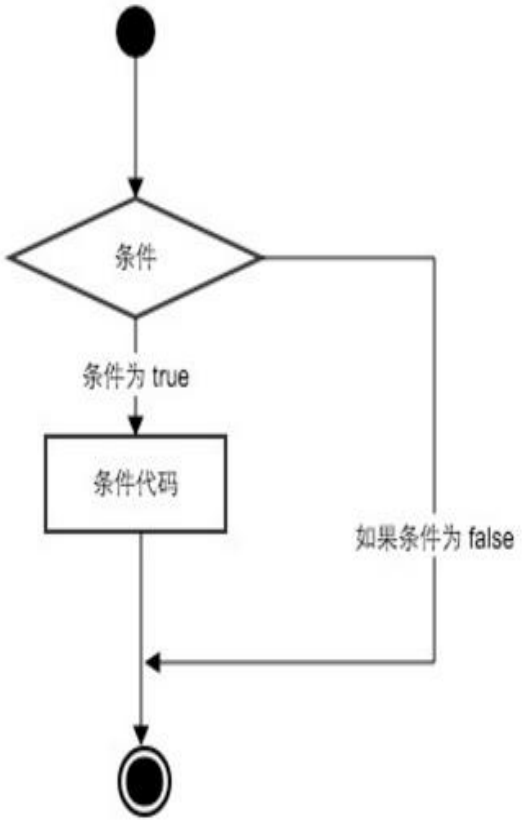
```
#!/usr/bin/python3

var1 = 100
if var1:
    print ("1 - if 表达式条件为 true")
    print (var1)

var2 = 0
if var2:
    print ("2 - if 表达式条件为 true")
    print (var2)
print ("Good bye!")
```

执行以上代码，输出结果为:

```
1 - if 表达式条件为 true
100
Good bye!
```



4. 控制语句

- 布尔变量的作用
  - True、False就是布尔变量。
  - 下面的值在作为布尔表达式的时候，会被解释器看作假（false）：
    - False： 标准值False
    - None： 标准值 None
    - 0： 所有类型的数字0（包括浮点型、长整型和其他类型）
    - “”： 如空字符串
    - () ： 空元组
    - [] ： 空列表
    - {}： 空的字典
  - 其它的都解释为真，包括原生的布尔值True。

4. 控制语句

❖ 循环语句 -- while

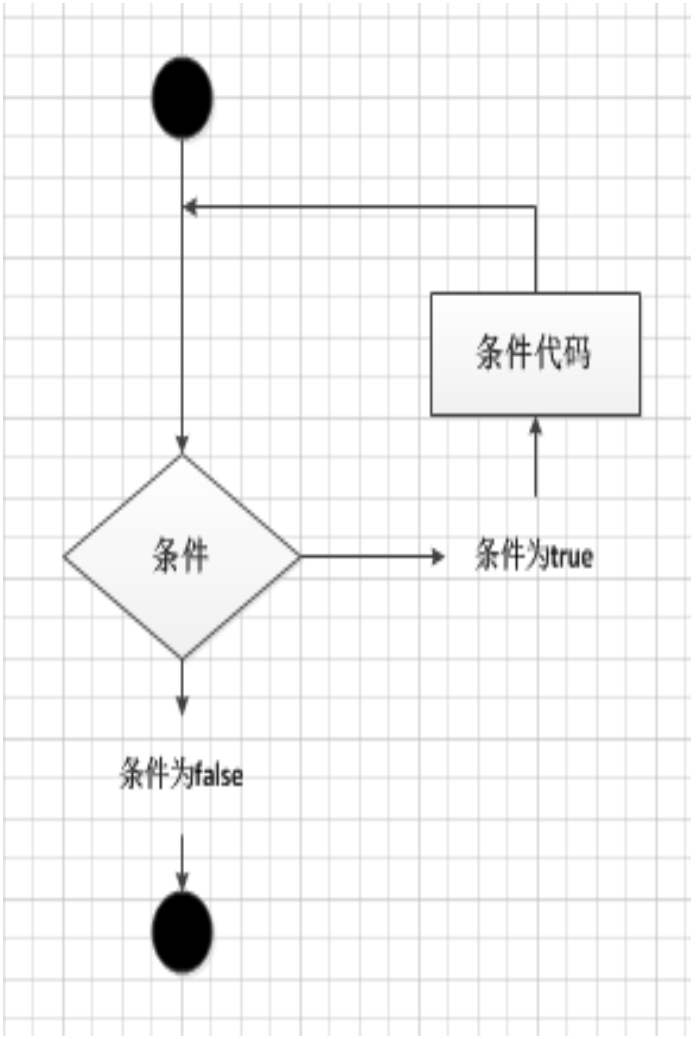
while 判断条件:
    语句

code

```
1  a = 1
2  while a < 10:
3   print (a)
4   a += 2
```

output

variables



4. 控制语句

❖ 循环语句 – for

for 语句

Python for循环可以遍历任何序列的项目，如一个列表或者一个字符串。

for循环的一般格式如下：

```
for <variable> in <sequence>:
    <statements>
else:
    <statements>
```

以下 for 实例中使用了 break 语句，break 语句用于跳出当前循环体：

实例

```
#!/usr/bin/python3

sites = ["Baidu", "Google","Runoob","Taobao"]
for site in sites:
    if site == "Runoob":
        print("菜鸟教程!")
        break
    print("循环数据 " + site)
else:
    print("没有循环数据!")
print("完成循环!")
```

执行脚本后，在循环到 "Runoob"时会跳出循环体：

```
循环数据 Baidu
循环数据 Google
菜鸟教程!
完成循环!
```

4. 控制语句

❖ 循环语句 – continue

continue语句被用来告诉Python跳过当前循环块中的剩余语句，然后继续进行下一轮循环。

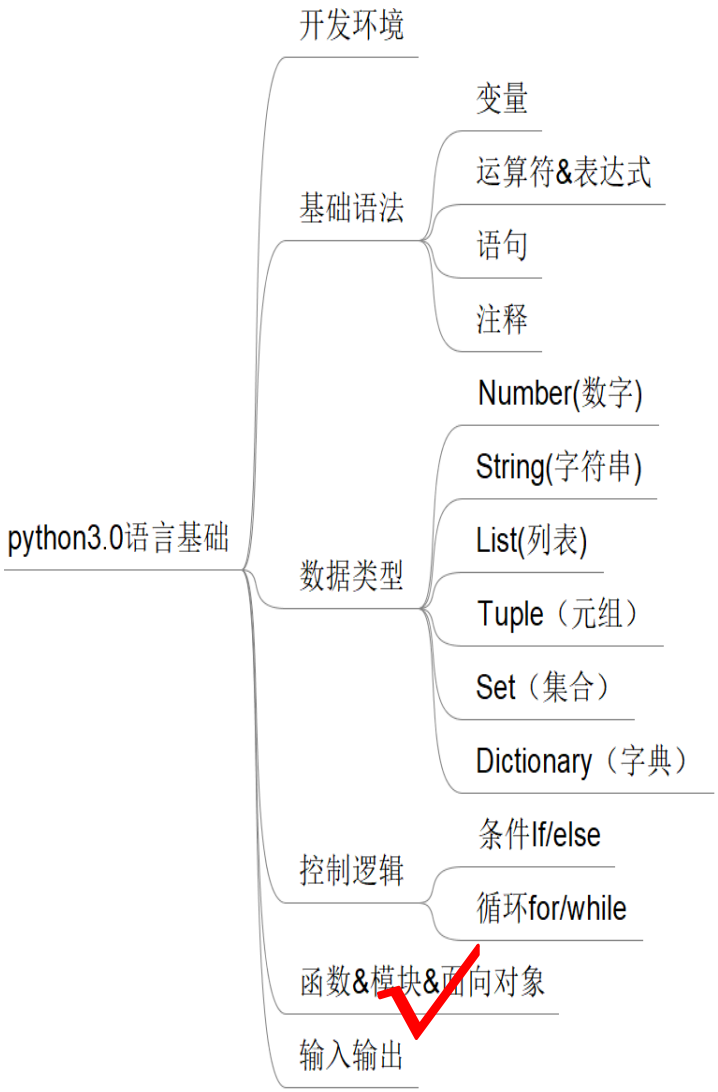
实例

```
#!/usr/bin/python3

for letter in 'Runoob':     # 第一个实例
   if letter == 'o':        # 字母为 o 时跳过输出
      continue
   print ('当前字母 :', letter)

var = 10                    # 第二个实例
while var > 0:
   var = var -1
   if var == 5:             # 变量为 5 时跳过输出
      continue
   print ('当前变量值 :', var)
print ("Good bye!")
```

内容提纲


python3.0语言基础
开发环境
基础语法
变量
运算符&表达式
语句
注释
数据类型
Number(数字)
String(字符串)
List(列表)
Tuple（元组）
Set（集合）
Dictionary（字典）
控制逻辑
条件If/else
循环for/while
函数&模块&面向对象
输入输出

5-1. 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。

•函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号 **()**。

•任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。

•函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。

•函数内容以冒号起始，并且缩进。

•**return [表达式]** 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

实例(Python 3.0+)

```
#!/usr/bin/python3

# 可写函数说明
def sum( arg1, arg2 ):
   # 返回2个参数的和."
   total = arg1 + arg2
   print ("函数内 : ", total)
   return total

# 调用sum函数
total = sum( 10, 20 )
print ("函数外 : ", total)
```

以上实例输出结果:

```
函数内 :  30
函数外 :  30
```

5-1. 函数（function）

- Python3内置了很多有用的函数，可以直接调用。
- 要调用一个函数，需要知道函数的名称和参数，比如求绝对值的函数abs，只有一个参数。可以直接从Python的官方网站查看文档：
- https://docs.python.org/3/library/functions.html
- 除了到Python官网查看文档，还可以在交互式命令行通过help(abs)查看abs函数的帮助信息。在交互模式下输入：

```
>>> help(abs)
Help on built-in function abs in module
builtins:
abs(x, /)
    Return the absolute value of the
argument.
```

5-2. 模块（Module）

模块是一个包含所有你定义的函数和变量的文件，其后缀名是.py。
模块可以被别的程序引入，以使用该模块中的函数等功能。

support.py 文件代码

```
#!/usr/bin/python3
# Filename: support.py

def print_func( par ):
    print ("Hello : ", par)
    return
```

test.py 引入 support 模块：

test.py 文件代码

```
#!/usr/bin/python3
# Filename: test.py

# 导入模块
import support

# 现在可以调用模块里包含的函数了
support.print_func("Runoob")
```

from ... import 语句

Python 的 from 语句让你从模块中导入一个指定的部分到当前命名空间中，语法如下：

```
from modname import name1[, name2[, ... nameN]]
```

例如，要导入模块 fibo 的 fib 函数，使用如下语句：

```
>>> from fibo import fib, fib2
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

这个声明不会把整个fibo模块导入到当前的命名空间中，它只会将fibo里的fib函数引入进来。

from ... import * 语句

把一个模块的所有内容全都导入到当前的命名空间也是可行的，只需使用如下声明：

```
from modname import *
```

这提供了一个简单的方法来导入一个模块中的所有项目。然而这种声明不该被过多地使用。

5-3. 包（Package）

- 包是一种管理 Python 模块命名空间的形式，采用"点模块名称"。
- 比如一个模块的名称是 A.B， 那么他表示一个包 A中的子模块 B 。
- 就好像使用模块的时候，你不用担心不同模块之间的全局变量相互影响一样，采用点模块名称这种形式也不用担心不同库之间的模块重名的情况。
- 目录只有包含一个叫做 __init__.py 的文件才会被认作是一个包

```
sound/                          顶层包
      __init__.py               初始化 sound 包
      formats/                  文件格式转换子包
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  声音效果子包
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  filters 子包
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

5-4. 面向对象编程

❖ 类和实例

- 面向对象最重要的概念就是类（Class）和实例（Instance），必须牢记类是抽象的模板，比如Student类，而实例是根据类创建出来的一个个具体的“对象”，每个对象都拥有相同的方法，但各自的数据可能不同。

```
class Animal():
    def __init__(self,name,weight):
        self.name = name
        self.weight = weight
    def print_weight(self):
        print("%s : %s" % (self.name,self.weight))

dog = Animal('hasky',90)
dog.print_weight() # hasky : 90
```

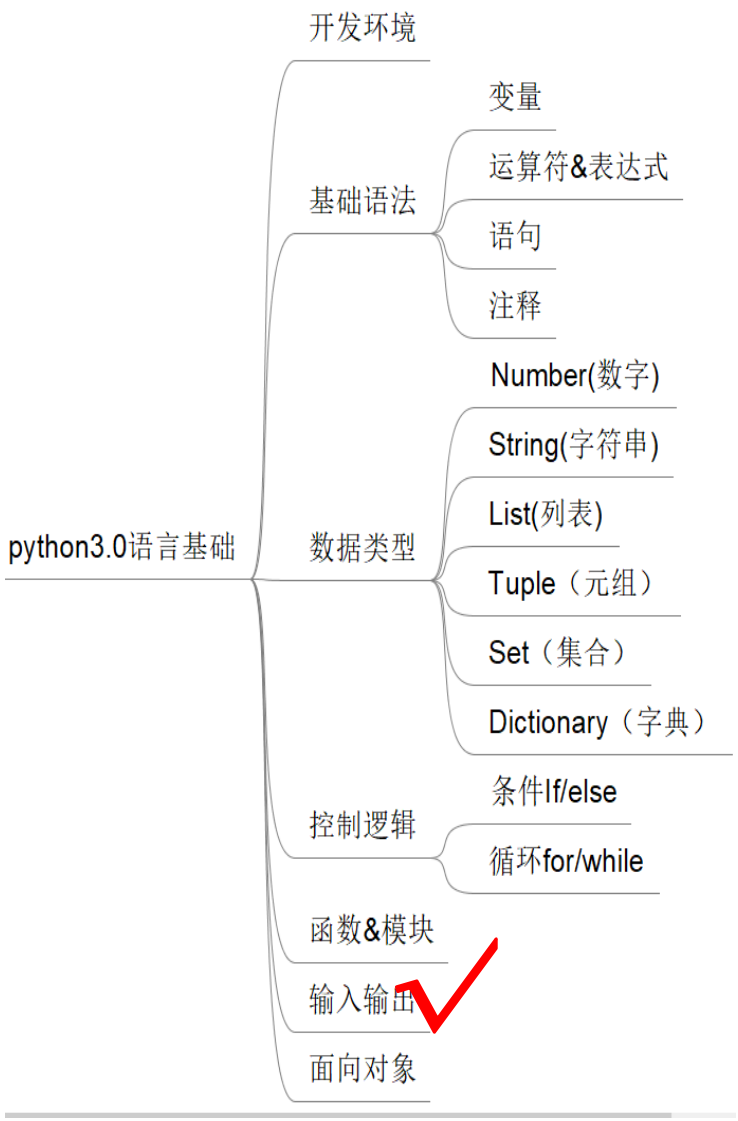
5-5. 典型的python文件结构

# Description	1. 起始行
”this is a test module”	2. 模块文档(字符串)
import sys import os	3. 模块导入
debug = True	4. 全局变量定义
class FooClass (object): ”Foo Class” pass	5. 类定义（若有）

def test():
”test function”
foo = FooClass()
if debug:
print ’ran test’
	6. 函数定义（若有）

if __name__ == ’__main__’ : test()	7. 主程序

内容提纲


python3.0语言基础
开发环境
基础语法
变量
运算符&表达式
语句
注释
数据类型
Number(数字)
String(字符串)
List(列表)
Tuple（元组）
Set（集合）
Dictionary（字典）
控制逻辑
条件If/else
循环for/while
函数&模块
输入输出
面向对象

6. 输入输出

- 打开文件
  - 方式一

```
f = open(“data.txt”, “r”, encoding='UTF-8')  # 设置文件对象
f.close()  # 关闭文件
```

  - 方式二

```
# 为了方便，避免忘记close掉这个文件对象，可以用下面这种方式替代
with open('data.txt', "r“, encoding='UTF-8') as f:  # 设置文件对象
    str = f.read()  # 可以是随便对文件的操作
```

6. 输入输出

❖ 读文件

- **1.简单的将文件读取到字符串中**
  - **读取文件最简单的方法是使用read()，将从文件中一次性读出所有内容，并赋值给1个字符串变量。**

```
f = open("data.txt","r")   #设置文件对象
str = f.read()     #将txt文件的所有内容读入到字符串str中
f.close()   #将文件关闭
```

6. 输入输出

- **2.按行读取整个文件**

```
#第一种方法
f = open("data.txt","r")   #设置文件对象
line = f.readline()
line = line[:-1]
while line:                #直到读取完文件
    line = f.readline()  #读取一行文件，包括换行符
    line = line[:-1]     #去掉换行符，也可以不去
f.close() #关闭文件
```

6. 输入输出

- **2.按行读取整个文件**

```
#第二种方法
data = []
for line in open("data.txt","r"): #设置文件对象并
读取每一行文件
    data.append(line)             #将每一行文件加
入到list中
```

6. 输入输出

- **2.按行读取整个文件**
  - **使用readlines()读取文件，需要通过循环访问readlines()返回列表中的元素。函数readlines()可一次性读取文件中多行数据。**

```
#第三种方法
f = open("data.txt","r")   #设置文件对象
data = f.readlines()  #直接将文件中按行读到list里，
效果与方法2一样
f.close()            #关闭文件
```

6. 输入输出

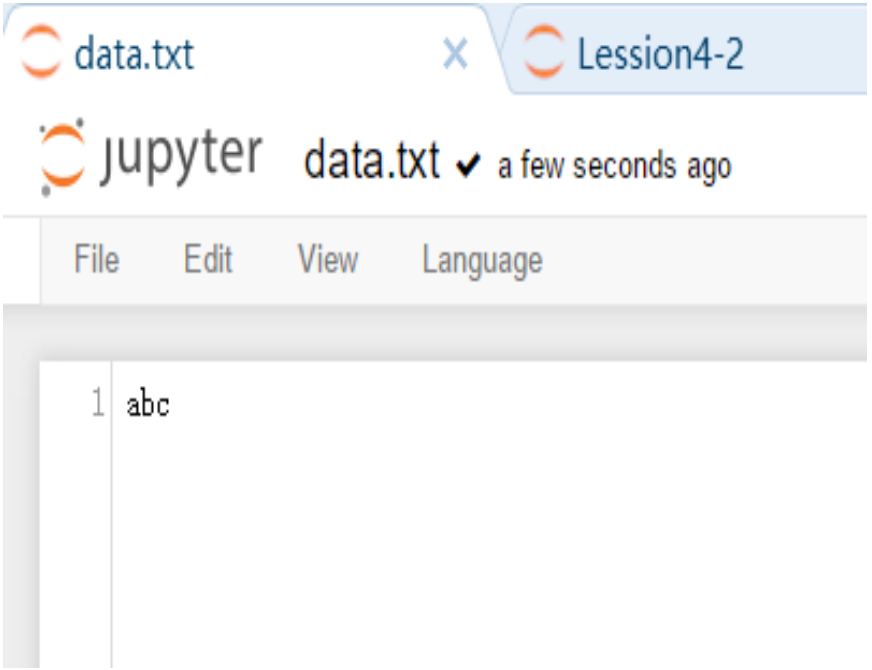
- 写文件
  - 1.简单的将字符串写入txt中

```
with open('data.txt','w') as f:    #设置文件对象
    f.write(str)                   #将字符串写入文件中
```

6. 输入输出

- 写文件
  - 2.列表写入文件

```
data = ['a','b','c']
#单层列表写入文件
with open("data.txt","w") as f:
    f.writelines(data)
```



6. 输入输出

❖ CSV文件

- 逗号分隔值（Comma-Separated Values，CSV，有时也称为字符分隔值，因为分隔字符也可以不是逗号），其文件以纯文本形式存储表格数据（数字和文本）。纯文本意味着该文件是一个字符序列，不含必须像二进制数字那样被解读的数据。
- CSV文件由任意数目的记录组成，记录间以某种换行符分隔；每条记录由字段组成，字段间的分隔符是其它字符或字符串，最常见的是逗号或制表符。
- CSV文件本身就是个纯文本文件，这种文件格式经常用来作为不同程序之间的数据交互的格式。

6. 输入输出

- CSV函数
  - 使用 CSV 函数需要导入csv库：import csv
    - csv.reader
    - csv.writer
    - csv.DictReader
    - csv.DictWriter

6. 输入输出

- **读取CSV文件**
  - **csv.reader(csvfile，dialect ='excel'，** fmtparams)**
  - **返回一个reader对象，它将迭代给定csvfile中的行。**

```
import csv
with open('some.csv', 'rb') as f:      # 采用b的方式处理可以省去很多问题
    reader = csv.reader(f)
    for row in reader:
        # do something with row, such as row[0],row[1]
```

  - **csv.writer(csvfile，dialect ='excel'，** fmtparams)**
  - **返回一个编写器对象，负责将用户的数据转换为给定的类文件对象上的分隔字符串。**

```
import csv
with open('some.csv', 'wb') as f:      # 采用b的方式处理可以省去很多问题
    writer = csv.writer(f)
    writer.writerows(someiterable)
```

6. 输入输出

- 字典方式
  - csv.DictReader(f，fieldnames = None，restkey = None，restval = None，dialect ='excel'，* args，** kwds)
  - 创建一个像常规阅读器一样操作的对象，但将读取的信息映射到一个dict，其键由可选的 fieldnames参数给出。

```
import csv
with open('names.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row['first_name'], row['last_name'])
```

  - csv.DictWriter(f，fieldnames，restval =''，extrasaction ='raise'，dialect ='excel'，* args，** kwds)
  - 创建一个像常规编写器一样操作的对象，但将字典映射到输出行。

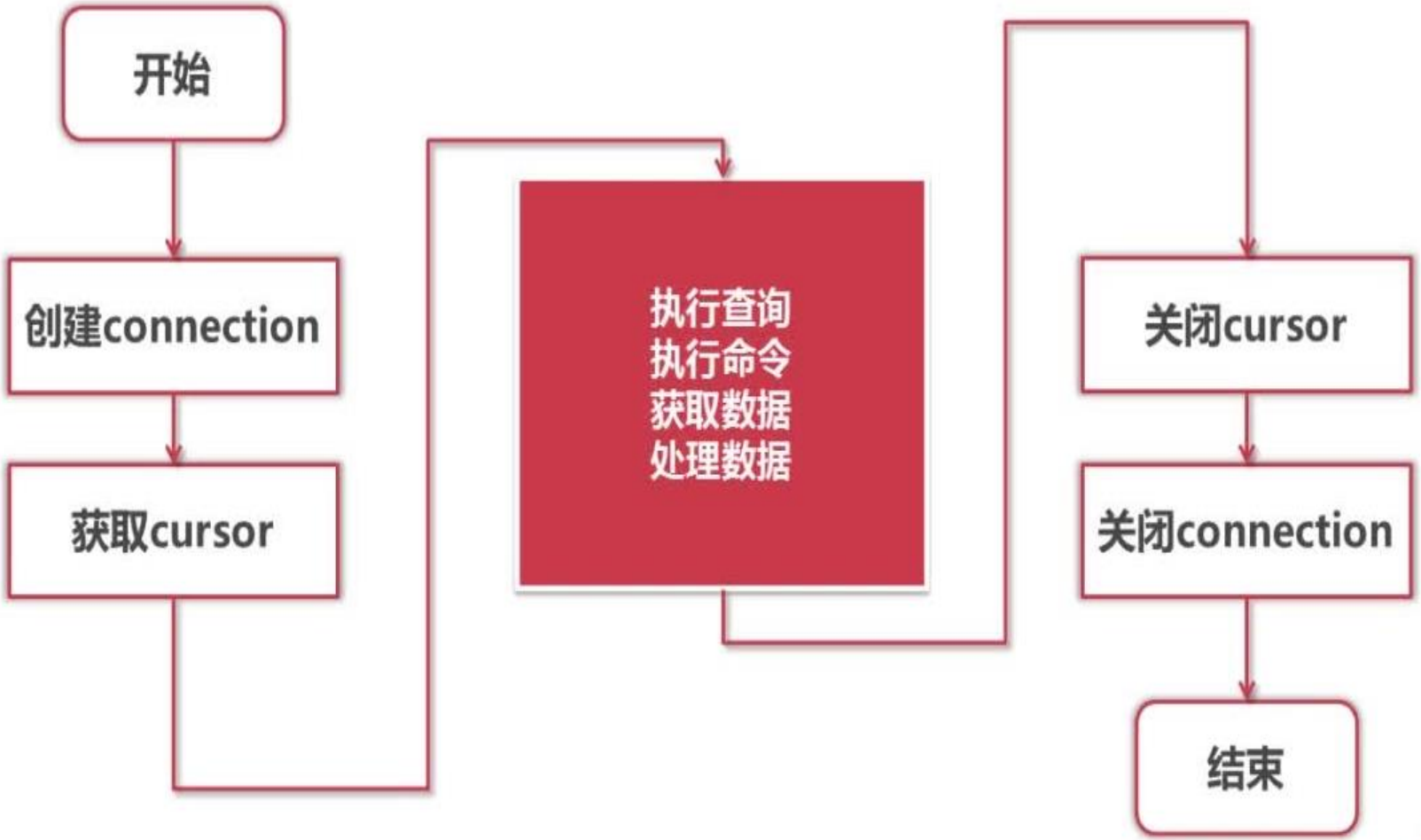
```
import csv
with open('names.csv', 'w') as csvfile:
    fieldnames = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile,
fieldnames=fieldnames)
```

6. 输入输出

- 数据库
  - 关系型数据库
    - MySQL
    - SQL-Server
    - SQLite
  - 非关系型数据库
    - MongoDB
    - Neo4j

6. 输入输出

- MySQL与Python的交互



6. 输入输出

MySQL与Python的交互

- **安装**
  pip3 install pymysql

- **使用**
  import pymysql

- **链接数据库**

```
Db = pymysql.Connect(host='127.0.0.1', port=3306, user='root',
password='123456', database='test08', charset='utf8’)
```

- **数据库游标**

```
cursor = db.cursor()
```

6. 输入输出

MySQL与Python的交互

- # 查询数据
- db.begin()
- cursor.execute("select * from students_info;")
- db.commit()

- # 获取所有数据
- print(cursor.fetchall())

- # 获取一个，根据下标取对应的数据
- print(cursor.fetchall()[0])

- # 注: 不能同时使用，因为游标会往后移动

6. 输入输出

- MySQL与Python的交互
  - # 插入数据

```
db.begin()
cursor.execute("insert into students_info values ('2000', '老李', '18', '男', '广东深圳', '1703', '89', '90', '81');")
db.commit()
```

  - # 更新数据

```
db.begin()
cursor.execute("update students_info set class='1807' where id=2000")
db.commit()
```

  - # 删除数据

```
db.begin()
cursor.execute("delete from students_info where id=2000")
db.commit()
```