

Python开发入门

NSD PYTHON1

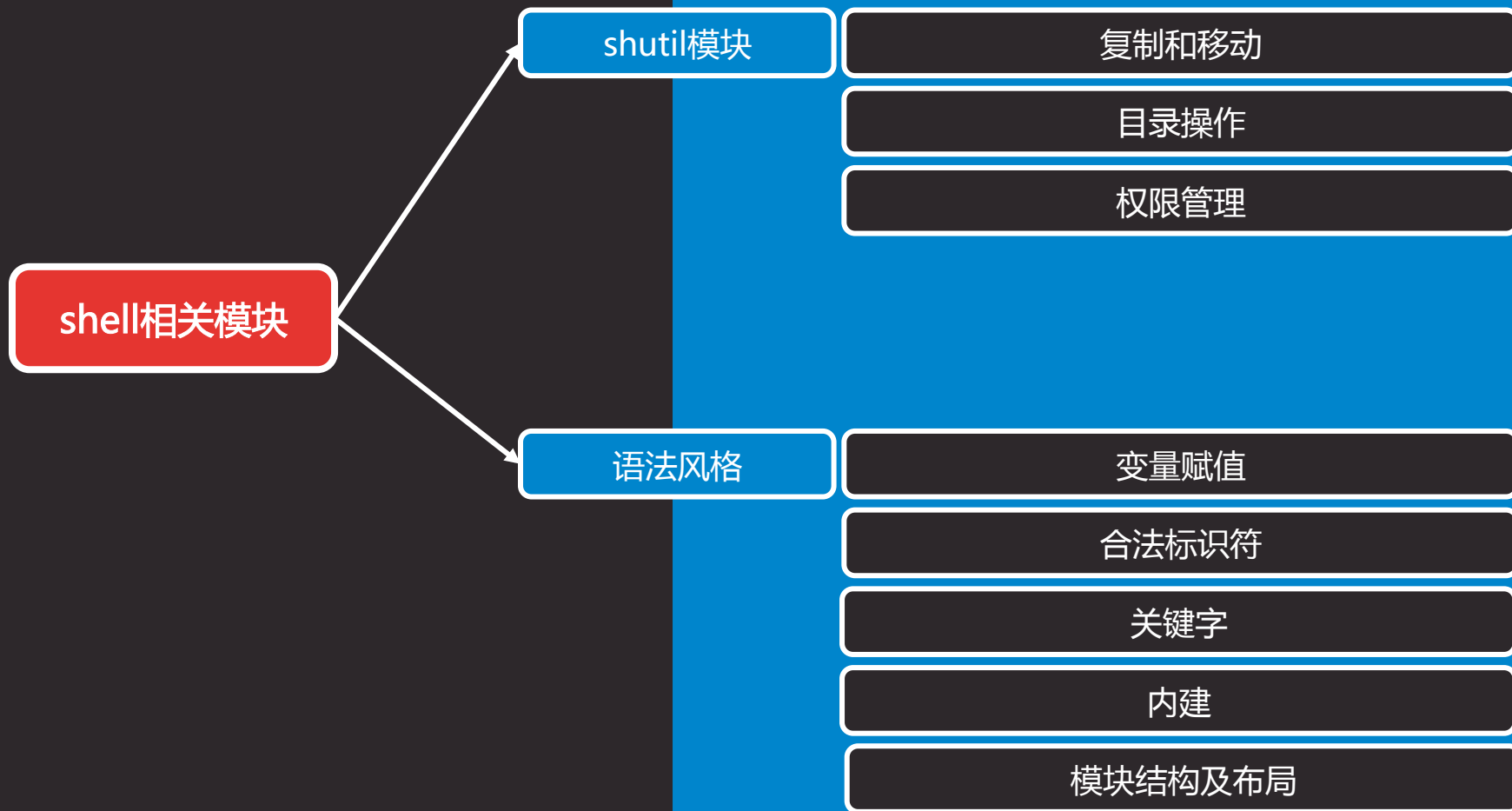
DAY04

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	shell相关模块
	10:30 ~ 11:20	
	11:30 ~ 12:00	字符串详解
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	列表和元组
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑



shell相关模块



shutil模块

复制和移动

- `shutil.copyfileobj(fsrc, fdst[, length])`
将类似文件的对象fsrc的内容复制到类似文件的对象fdst。
- `shutil.copyfile(src, dst, *, follow_symlinks=True)`
将名为src的文件的内容（无元数据）复制到名为dst的文件，然后返回dst。



复制和移动（续1）

- `shutil.copy(src, dst, *, follow_symlinks=True)`
将文件src复制到文件或目录dst。src和dst应为字符串。如果dst指定目录，则文件将使用src的基本文件名复制到dst中。返回新创建的文件的完整路径。
- `shutil.copy2(src, dst, *, follow_symlinks=True)`
与copy()相同，但copy2()也尝试保留所有文件元数据。
- `shutil.move(src, dst, copy_function=copy2)`
递归地将文件或目录（src）移动到另一个位置（dst），并返回目标。



目录操作

- `shutil.copytree(src, dst, symlinks=False, ignore=None, copy_function=copy2, ignore_dangling_symlinks=False)`
递归地复制以src为根的整个目录树，返回目标目录。由dst命名的目标目录不能已经存在。
- `shutil.rmtree(path, ignore_errors=False, onerror=None)`
删除整个目录树；路径必须指向目录（而不是指向目录的符号链接）。



权限管理

- `shutil.copymode(src, dst, *, follow_symlinks=True)`
将权限位从src复制到dst。文件内容，所有者和组不受影响。src和dst是以字符串形式给出的路径名称。
- `shutil.copystat(src, dst, *, follow_symlinks=True)`
将权限位，最后访问时间，上次修改时间和标志从src复制到dst。
- `shutil.chown(path, user=None, group=None)`
更改给定路径的所有者用户和/或组



语法风格

变量赋值

- python支持链式多重赋值

`x = y = 10`

- 另一种将多个变量同时赋值的方法称为多元赋值，采用这种方式赋值时，等号两边的对象都是元组

`a, b = 10, 20`



合法标识符

- python标识符字符串规则和其他大部分用C编写的高级语言相似
- 第一个字符必须是字母或下划线（_）
- 剩下的字符可以是字母和数字或下划线
- 大小写敏感



关键字

- 和其他的高级语言一样，python也拥有一些被称作关键这字的保留字符
- 任何语言的关键字应该保持相对的稳定，但是因为python是一门不断成长和进化的语言，其关键字偶尔会更新
- 关键字列表和iskeyword()函数都放入了keyword模块以便查阅



内建

- 除了关键字之外，python还有可以在任何一级代码使用的“内建”的名字集合，这些名字可以由解释器设置或使用
- 虽然built-in不是关键字，但是应该把它当作“系统保留字”
- 保留的常量如：True、False、None



模块结构及布局

- 编写程序时，应该建立一种统一且容易阅读的结构，并将它应用到每一个文件中

```
#!/usr/bin/env python      #起始行
    "this is a test module"  #模块文档字符串
import sys                 #导入模块
import os
debug = True               #全局变量声明
class FooClass(object):    #类定义
    'Foo class'
    pass
def test():                #函数定义
    "test function"
    foo = FooClass()
if __name__ == '__main__': #程序主体
    test()
```

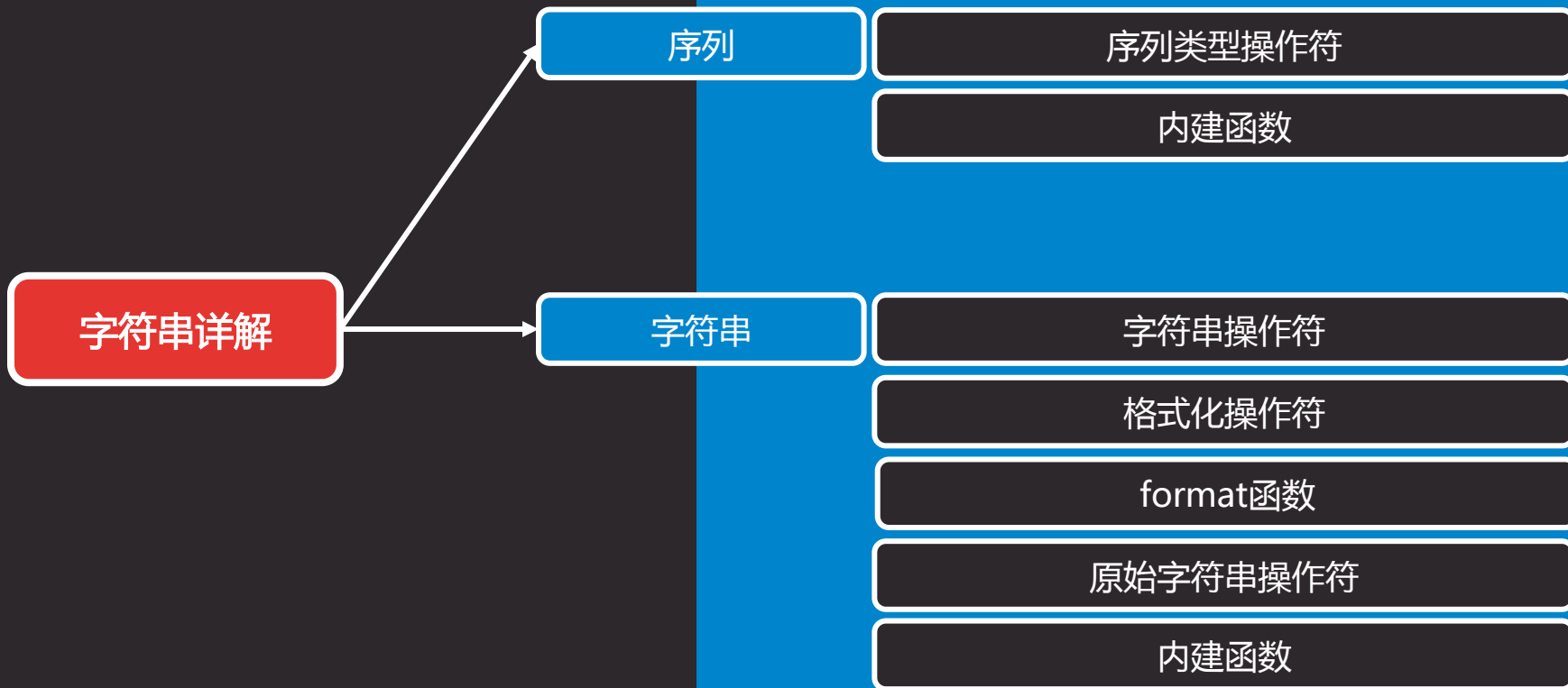


案例1：创建文件

1. 编写一个程序，要求用户输入文件名
2. 如果文件已存在，要求用户重新输入
3. 提示用户输入数据，每行数据先写到列表中
4. 将列表数据写入到用户输入的文件名中



字符串详解



序列



序列类型操作符

序列操作符	作 用
seq[ind]	获得下标为ind的元素
seq[ind1:ind2]	获得下标从ind1到ind2间的元素集合
seq * expr	序列重复expr次
seq1 + seq2	连接序列seq1和seq2
obj in seq	判断obj元素是否包含在seq中
obj not in seq	判断obj元素是否不包含在seq中



内建函数

函 数	含 义
list(iter)	把可迭代对象转换为列表
str(obj)	把obj对象转换成字符串
tuple(iter)	把一个可迭代对象转换成一个元组对象

```
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
>>> list(('hello', 'world'))
['hello', 'world']
>>> str(['hello', 'world'])
"['hello', 'world']"
```



内建函数（续1）

- `len(seq)` : 返回seq的长度
- `max(iter, key=None)` : 返回iter中的最大值
- `enumerate` : 接受一个可迭代对象作为参数，返回一个`enumerate`对象

```
>>> alist = ['hello', 'world']
>>> for i, j in enumerate(alist):
...     print('index %d: %s' % (i, j))
...
index 0: hello
index 1: world
```



内建函数（续2）

- `reversed(seq)`：接受一个序列作为参数,返回一个以逆序访问的迭代器
- `sorted(iter)`：接受一个可迭代对象作为参数，返回一个有序的列表



字符串



字符串操作符

- 比较操作符：字符串大小按ASCII码值大小进行比较
- 切片操作符：[], [:], [::]
- 成员关系操作符：in、not in

```
>>> py_str = 'Hello World!'
```

```
>>> py_str[::-2]
```

```
'HloWrD'
```

```
>>> py_str[::-1]
```

```
'!dlroW olleH'
```



案例2：检查标识符

1. 程序接受用户输入
2. 判断用户输入的标识符是否合法
3. 用户输入的标识符不能使用关键字
4. 有不合法字符，需要指明第几个字符不合法



格式化操作符

- 字符串可以使用格式化符号来表示特定含义

格式化字符	转换方式
%c	转换成字符
%s	优先用str()函数进行字符串转换
%d / %i	转成有符号十进制数
%o	转成无符号八进制数
%e / %E	转成科学计数法
%f / %F	转成浮点数



格式化操作符（续1）

- 字符串可以使用格式化符号来表示特定含义

辅助指令	作 用
*	定义宽度或者小数点精度
-	左对齐
+	在正数前面显示加号
<sp>	在正数前面显示空格
#	在八进制数前面显示零0，在十六进制前面显示'0x'或者'0X'
0	显示的数字前面填充0而不是默认的空格



format函数

- 使用位置参数
 - 'my name is {} ,age {}'.format('hoho',18)
- 使用关键字参数
 - 'my name is {name},age is {age}'.format({'name':'bob', 'age':23})
- 填充与格式化
 - {: [填充字符][对齐方式 <^>][宽度]}
- 使用索引
 - 'name is {0[0]} age is {0[1]}'.format(['bob', 23])



案例3：创建用户

1. 编写一个程序，实现创建用户的功能
2. 提示用户输入用户名
3. 随机生成8位密码
4. 创建用户并设置密码
5. 将用户相关信息写入指定文件



原始字符串操作符

- 原始字符串操作符是为了对付那些在字符串中出现的特殊字符
- 在原始字符串里，所有的字符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

```
>>> winPath = "c:\windows\temp"  
>>> print(winPath)  
c:\windows    emp  
>>> newPath = r"c:\windows\temp"  
>>> print(newPath)  
c:\windows\temp
```



案例4：格式化输出

1. 提示用户输入（多行）数据
2. 假定屏幕的宽度为50，用户输入的多行数据如下显示（文本内容居中）：

```
+*****+
+               hello world      +
+               great work!      +
+*****+
```



内建函数

- `string.capitalize()` : 把字符串的第一个字符大写
- `string.center(width)` : 返回一个原字符串居中, 并使用空格填充至长度width 的新字符串
- `string.count(str, beg=0,end=len(string))` : 返回str 在string里面出现的次数, 如果beg或者end指定则返回指定范围内str出现的次数



内建函数（续1）

- `string.endswith(obj, beg=0, end=len(string))` : 检查字符串是否以obj结束，如果beg或者end指定则检查指定的范围内是否以obj结束，如果是，返回True，否则返回False
- `string.islower()` : 如果string中包含至少一个区分大小写的字符，并且所有这些字符都是小写，则返回True，否则返回False

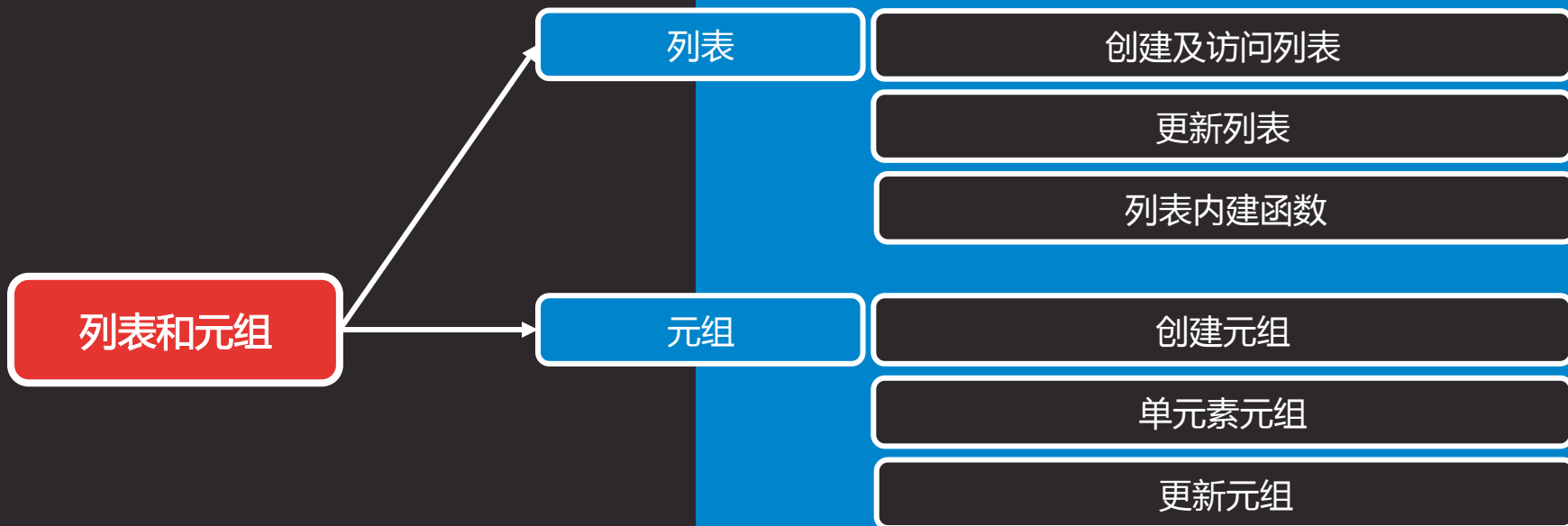


内建函数（续2）

- `string.strip()`：删除string 字符串两端的空白
- `string.upper()`：转换string 中的小写字母为大写
- `string.split(str="", num=string.count(str))`：以str 为分隔符切片string，如果num有指定值，则仅分隔num个子字符串



列表和元组



列表

创建及访问列表

- 列表是有序、可变的数据类型
- 列表中可以包含不同类型的对象
- 列表可以由[]或工厂函数创建
- 支持下标及切片操作



更新列表

- 通过下标只能更新值，不能使用标添加新值

```
>>> alist = [10, 35, 20, 80]  
>>> alist[-1] = 100  
>>> alist[1:3] = [30, 50, 80]
```



列表内建函数

列表方法	操 作
<code>list.append(obj)</code>	向列表中添加一个对象obj
<code>list.count(obj)</code>	返回一个对象obj 在列表中出现的次数
<code>list.extend(seq)</code>	把序列seq的内容添加到列表中
<code>list.index(obj)</code>	返回obj对象的下标
<code>list.insert(index, obj)</code>	在索引量为index 的位置插入对象obj
<code>list.reverse()</code>	原地翻转列表
<code>list.sort()</code>	排序



元组

创建元组

- 通过()或工厂函数tuple()创建元组
- 元组是有序的、不可变类型
- 与列表类似，作用于列表的操作，绝大多数也可以作用于元组



单元素元组

- 如果一个元组中只有一个元素，那么创建该元组的时候，需要加上一个逗号

```
>>> atuple = ('hello')
>>> print(atuple)
hello
>>> type(atuple)
<class 'str'>
>>> atuple = ('hello',)
>>> print(atuple)
('hello',)
>>> type(atuple)
<class 'tuple'>
```



“更新” 元组

- 虽然元组本身是不可变的，但是因为它同时属于容器类型，也就意味着元组的某一个元素是可变的容器类型，那么这个元素中的项目仍然可变

```
>>> atuple = ('bob', ['boy', 23])
```

```
>>> atuple[-1] = ['boy', 22]
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```

```
>>> atuple[-1][-1] = 22
```

```
>>> atuple[-1].append(175)
```

```
>>> atuple
```

```
('bob', ['boy', 22, 175])
```



案例5：用列表构建栈结构

1. 栈是一个后进先出的结构
2. 编写一个程序，用列表实现栈结构
3. 需要支持压栈、出栈、查询功能



总结和答疑
