Tedu.cn
达内教育

Python开发进阶

NSD PYTHON2

DAY03

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	OOP基础
	10:30 ~ 11:20	OOP进阶
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	re模块
	15:00 ~ 15:50	
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑

OOP基础

Tedu.cn
达内教育

OOP简介

基本概念

- 类(Class)：用来描述具有相同的属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。对象是类的实例。
- 实例化：创建一个类的实例，类的具体对象。
- 方法：类中定义的函数。
- 对象：通过类定义的数据结构实例。对象包括两个数据成员（类变量和实例变量）和方法。

知识讲解

创建类

- 使用 class 语句来创建一个新类，class 之后为类的名称并以冒号结尾
- 类名建议使用驼峰形式

```
class BearToy:
    pass
```

知识讲解

创建实例

- 类是蓝图，实例是根据蓝图创建出来的具体对象

```
tidy = BearToy()
```

知识讲解

Tedu.cn
达内教育

绑定方法

构造器方法

- 当实例化类的对象是，构造器方法默认自动调用
- 实例本身作为第一个参数，传递给self

知识讲解

```
class BearToy:
    __init__(self, size, color):
        self.size = size
        slef.color = color

if __name__ == '__main__':
    tidy = BearToy('small', 'orange')
```

其他绑定方法

- 类中定义的方法需要绑定在具体的实例，由实例调用
- 实例方法需要明确调用

知识讲解

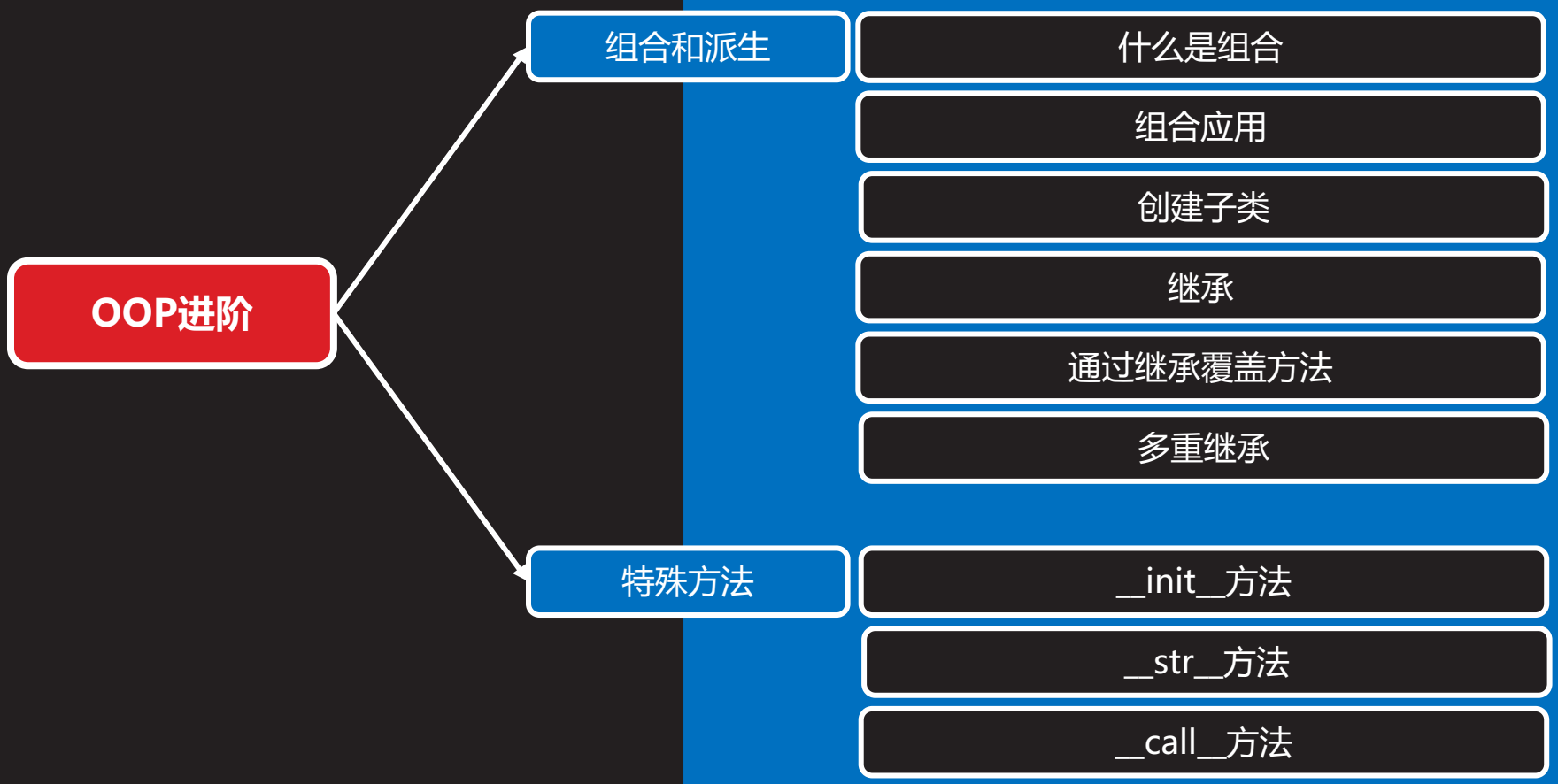
```
class BearToy:
    def __init__(self, size, color):
        self.size = size
        self.color = color

    def speak(self):
        print('hahaha')

if __name__ == '__main__':
    tidy = BearToy('small', 'orange')
    tidy.speak()
```

案例1：编写游戏人物

1. 创建游戏角色类
2. 游戏人物角色拥有名字、武器等属性
3. 游戏人物具有攻击和行走的方法
4. 武器通过武器类实现

课堂练习

OOP进阶


OOP进阶
组合和派生
什么是组合
组合应用
创建子类
继承
通过继承覆盖方法
多重继承
特殊方法
__init__方法
__str__方法
__call__方法

Tedu.cn
达内教育

组合和派生

什么是组合

- 类被定义后，目标就是要把它当成一个模块来使用，并把这些对象嵌入到你的代码中去
- 组合就是让不同的类混合并加入到其它类中来增加功能和代码重用性
- 可以在一个大点的类中创建其它类的实例，实现一些其它属性和方法来增强对原来的类对象

知识讲解

组合应用

- 两个类明显不同
- 一个类是另一个类的组件

知识讲解

```
class Manufacture:
    def __init__(self, phone, email):
        self.phone = phone
        self.email = email

class BearToy:
    def __init__(self, size, color, phone, email):
        self.size = size
        self.color = color
        self.vendor = Manufacture(phone, email)
```

创建子类

知识讲解

- 当类之间有显著的不同，并且较小的类是较大的类所需要的组件时组合表现得很好；但当设计“相同的类但有一些不同的功能”时，派生就是一个更加合理的选择了
- OOP 的更强大方面之一是能够使用一个已经定义好的类，扩展它或者对其进行修改，而不会影响系统中使用现存类的其它代码片段
- OOD（面向对象设计）允许类特征在子孙类或子类中进行继承

创建子类（续1）

- 创建子类只需要在圆括号中写明从哪个父类继承即可

```
class BearToy:
    def __init__(self, size, color):
        self.size = size
        self.color = color
    ... ...

class NewBearToy:
    pass
```

知识讲解

继承

- 继承描述了基类的属性如何“遗传”给派生类
- 子类可以继承它的基类的任何属性，不管是数据属性还是方法

```
class BearToy:
    def __init__(self, size, color):
        self.size = size
        self.color = color
    … …

class NewBearToy:
    pass

if __name__ == '__main__':
    tidy = NewBearToy('small', 'orange')
    tidy.speak()
```

知识讲解

通过继承覆盖方法

- 如果子类中有和父类同名的方法，父类方法将被覆盖
- 如果需要访问父类的方法，则要调用一个未绑定的父类方法，明确给出子类的实例

```
class BearToy:
    def __init__(self, size, color, phone, email):
        self.size = size
        self.color = color
        self.vendor = Manufacture(phone, email)
    … …

class NewBearToy(BearToy):
    def __init__(self, size, color, phone, email, date):
        super(NewBearToy, self).__init__(size, color, phone, email)
        self.date = date
```

知识讲解

多重继承

- python允许多重继承，即一个类可以是多个父类的子类，子类可以拥有所有父类的属性

```
>>> class A:
...     def foo(self):
...         print('foo method’)
>>> class B:
...     def bar(self):
...         print('bar method’)
>>> class C(A, B):
...     pass
>>> c = C()
>>> c.foo()
foo method
>>> c.bar()
bar method
```

知识讲解

Tedu.cn
达内教育

特殊方法

__init__方法

- 实例化类实例时默认会调用的方法

```
class BearToy:
    __init__(self, size, color):
        self.size = size
        slef.color = color

if __name__ == '__main__':
    tidy = BearToy('small', 'orange')
```

知识讲解

__str__方法

- 打印/显示实例时调用方法
- 返回字符串

```
class BearToy:
    __init__(self, size, color):
        self.size = size
        slef.color = color

    def __str__(self):
        return '<Bear: %s %s>' % (self.size, self.color)

if __name__ == '__main__':
    tidy = BearToy('small', 'orange')
    print(tidy)
```

知识讲解

__call__方法

- 用于创建可调用的实例

```
class BearToy:
    __init__(self, size, color):
        self.size = size
        slef.color = color

    def __call__(self):
        print('I am a %s bear' % self.size)

if __name__ == '__main__':
    tidy = BearToy('small', 'orange')
    print(tidy)
```

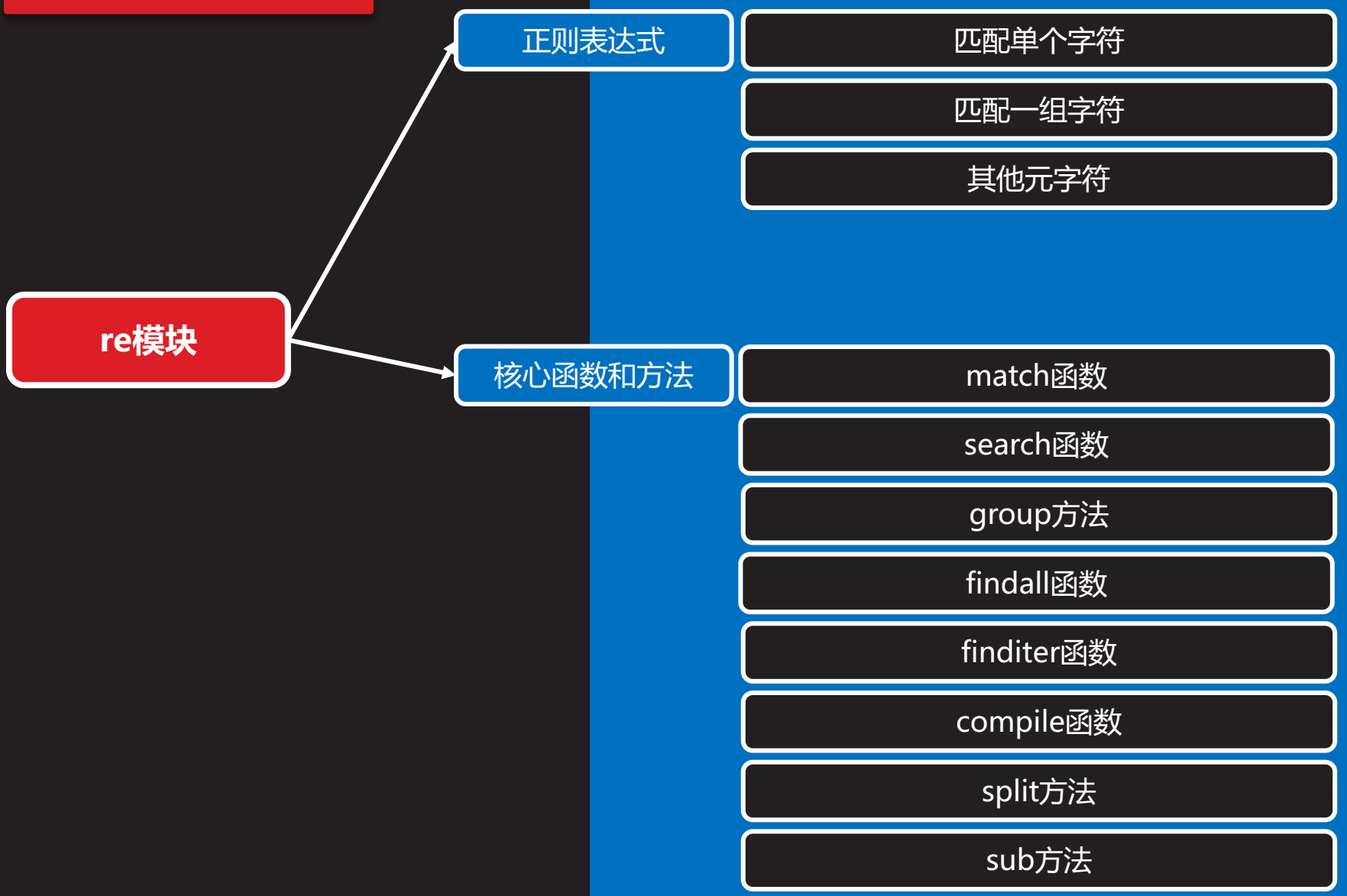
知识讲解

案例2：出版商程序

1. 为出版商编写一个Book类
2. Book类有书名、作者、页数等属性
3. 打印实例时，输出书名
4. 调用实例时，显示该书由哪个作者编写

课堂练习

re模块


re模块
正则表达式
匹配单个字符
匹配一组字符
其他元字符
核心函数和方法
match函数
search函数
group方法
findall函数
finditer函数
compile函数
split方法
sub方法

Tedu.cn
达内教育

正则表达式

匹配单个字符

知识讲解

记号	说 明
.	匹配任意字符（换行符除外）
[...x-y...]	匹配字符组里的任意字符
[^...x-y...]	匹配不在字符组里的任意字符
\d	匹配任意数字，与[0-9]同义
\w	匹配任意数字字母字符，与[0-9a-zA-Z_]同义
\s	匹配空白字符，与[\r\v\f\t\n]同义

匹配一组字符

知识讲解

记号	说 明
literal	匹配字符串的值
re1|re2	匹配正则表达式re1或re2
*	匹配前面出现的正则表达式零次或多次
+	匹配前面出现的正则表达式一次或多次
?	匹配前面出现的正则表达式零次或一次
{M, N}	匹配前面出现的正则表达式至少M次最多N次

其他元字符

知识讲解

记号	说 明
^	匹配字符串的开始
$	匹配字符串的结尾
\b	匹配单词的边界
()	对正则表达式分组
\nn	匹配已保存的子组

Tedu.cn
达内教育

核心函数和方法

match函数

- 尝试用正则表达式模式从字符串的开头匹配，如果匹配成功，则返回一个匹配对象；否则返回None

```
>>> import re
>>> m = re.match('foo', 'food')          #成功匹配
>>> print(m)
<_sre.SRE_Match object; span=(0, 3), match='foo'>
>>>
>>> m = re.match(‘foo’, ‘seafood’)      #未能匹配
>>> print(m)
None
```

知识讲解

search函数

- 在字符串中查找正则表达式模式的第一次出现，如果匹配成功，则返回一个匹配对象；否则返回None

```
>>> import re
>>> m = re.search('foo', 'food')
>>> print(m)
<_sre.SRE_Match object; span=(0, 3), match='foo'>
>>>
>>> m = re.search('foo', 'seafood')    #可以匹配在字符中间的模式
>>> print(m)
<_sre.SRE_Match object; span=(3, 6), match='foo'>
```

知识讲解

group方法

- 使用match或search匹配成功后，返回的匹配对象可以通过group方法获得匹配内容

```
>>> import re
>>> m = re.match('foo', 'food')
>>> print(m.group())
foo

>>> m = re.search('foo', 'seafood')
>>> m.group()
'foo'
```

知识讲解

findall函数

- 在字符串中查找正则表达式模式的所有（非重复）出现；返回一个匹配对象的列表

```
>>> import re
>>> m = re.search('foo', 'seafood is food')
>>> print(m.group())    #search只匹配模式的第一次出现
foo
>>>
>>> m = re.findall(‘foo’, ‘seafood is food’)  #获得全部的匹配项
>>> print(m)
['foo', 'foo']
```

知识讲解

finditer函数

- 和findall()函数有相同的功能，但返回的不是列表而是迭代器；对于每个匹配，该迭代器返回一个匹配对象

```
>>> import re
>>> m = re.finditer('foo', 'seafood is food')
>>> for item in m:
...     print(item.group())
...
foo
foo
```

知识讲解

compile函数

- 对正则表达式模式进行编译，返回一个正则表达式对象
- 不是必须要用这种方式，但是在大量匹配的情况下，可以提升效率

```
>>> import re
>>> patt = re.compile('foo')
>>> m = patt.match('food')
>>> print(m.group())
foo
```

知识讲解

split方法

- 根据正则表达式中的分隔符把字符分割为一个列表，并返回成功匹配的列表
- 字符串也有类似的方法，但是正则表达式更加灵活

```
>>> import re      #使用 . 和 - 作为字符串的分隔符
>>> mylist = re.split('\.|-', 'hello-world.data')
>>> print(mylist)
['hello', 'world', 'data']
```

知识讲解

sub方法

- 把字符串中所有匹配正则表达式的地方替换成新的字符串

知识讲解

```
>>> import re
>>> m = re.sub('X', 'Mr. Smith', 'attn: X\nDear X')
>>> print(m)
attn: Mr. Smith
Dear Mr. Smith
```

案例3：分析apache访问日志

- 编写一个apche日志分析脚本
  1. 统计每个客户端访问apache服务器的次数
  2. 将统计信息通过字典的方式显示出来
  3. 分别统计客户端是Firefox和MSIE的访问次数
  4. 分别使用函数式编程和面向对象编程的方式实现

课堂练习

Tedu.cn
达内教育

总结和答疑