



Python开发入门

NSD PYTHON1

DAY01

内容

上午	09:00 ~ 09:30	Python概述
	09:30 ~ 10:20	环境准备
	10:30 ~ 11:20	
	11:30 ~ 12:00	使用git
下午	14:00 ~ 14:50	
	15:00 ~ 15:50	Python起步
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑



python概述

python概述

python简介

Python起源

Python版本

Python的特点

Python简介

python起源

- 贵铎·范·罗萨姆 (Guido van Rossum) 于1989年底始创了python
- 1991年初，python发布了第一个公开发行版
- 为了更好的完成荷兰的CWI (国家数学和计算机科学研究院) 的一个研究项目而创建

知识讲解



Python版本

知识讲解

- Python2.x
 - 目前所有系统默认安装的版本
- Python3.x
 - 2009年2月13日发布
 - 在语法和功能上有较大调整
 - Python的发展趋势



Python的特点

知识讲解

- 高级：有高级的数据结构，缩短开发时间与代码量
- 面向对象：为数据和逻辑相分离的结构化和过程化编程添加了新的活力
- 可升级：提供了基本的开发模块，可以在它上面开发软件，实现代码的重用
- 可扩展：通过将其分离为多个文件或模块加以组织管理



Python的特点（续1）

知识讲解

- 可移植性：python是用C写的，又由于C的可移植性，使得python可以运行在任何带有ANSI C编译器的平台上
- 易学：python关键字少、结构简单、语法清晰
- 易读：没有其他语言通常用来访问变量、定义代码块和进行模式匹配的命令式符号
- 内存管理器：内存管理是由python解释器负责的



环境准备

环境准备

安装与配置

获取python3源码

安装python3

设置环境变量

设置pycharm

安装与配置

获取python3源码

- 官方站点
 - <http://www.python.org>
- 选择正确的系统
- 选择正确的版本

知识讲解



安装python3

知识讲解

- 安装依赖包

```
# yum install -y gcc gcc-c++ zlib-devel openssl-devel readline-devel  
libffi-devel sqlite-devel tcl-devel tk-devel
```

- 安装python3

```
# tar xzf Python-3.6.4.tar.gz  
# cd Python-3.6.4  
# ./configure --prefix=/usr/local  
# make && make install
```



设置pycharm

知识讲解

- Pycharm是由JetBrains打造的一款Python IDE
- 支持的功能有：
 - 调试、语法高亮
 - Project管理、代码跳转
 - 智能提示、自动完成
 - 单元测试、版本控制
- 下载地址：
<https://www.jetbrains.com/pycharm/download>
- 分为收费的专业版和免费的社区版



案例1：准备python开发环境

课堂练习

1. 下载最新版本的python3
2. 下载pycharm社区版
3. 安装python3，使其支持Tab键补全
4. 配置pycharm，使其符合自己的习惯



使用git

使用git

本地操作

Git简介

安装及配置

Git工作流程

工作区、暂存区和版本库

创建仓库

添加文件到暂存区

确认至仓库

删除跟踪文件

搭建本地gitlab服务器

初始化gitlab服务器

添加gitlab项目

创建群组

创建项目

创建用户

用户管理

使用远程服务器

本地操作

Git简介

- Git是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。
- Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。
- Git 与常用的版本控制工具 CVS, Subversion 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。

知识讲解



安装及配置

- Git安装后需配置用户相关信息

```
[root@localhost ~]# yum install -y git  
[root@localhost ~]# git config --global user.name "Mr.Zhang"  
[root@localhost ~]# git config --global user.email "zhangzg@tedu.cn"  
[root@localhost ~]# git config --global core.editor vim  
[root@localhost ~]# git config --list  
[root@localhost ~]# cat ~/.gitconfig
```

知识讲解



案例2：配置git

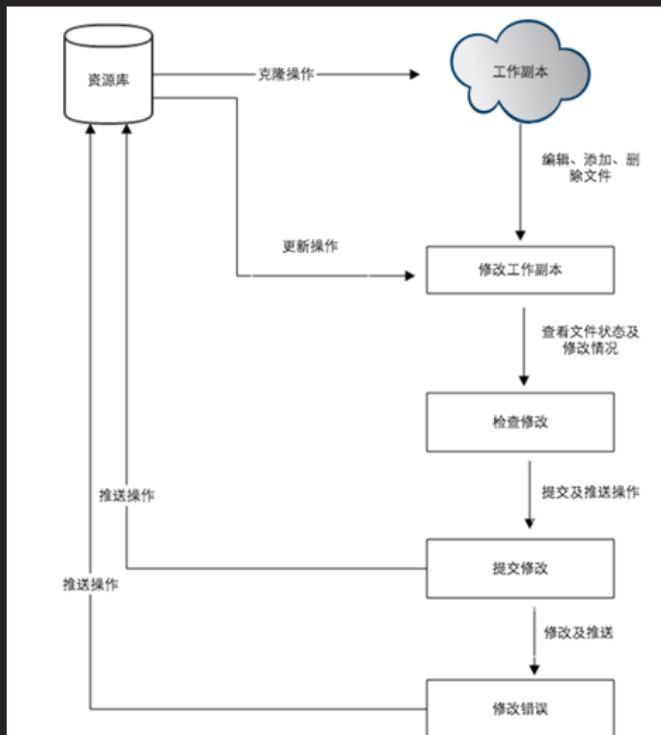
课堂练习

1. 安装git版本控制软件
2. 设置用户信息，如用户名、email等
3. 设置默认编辑器为vim
4. 查看用户配置



Git工作流程

知识讲解



工作区、暂存区和版本库

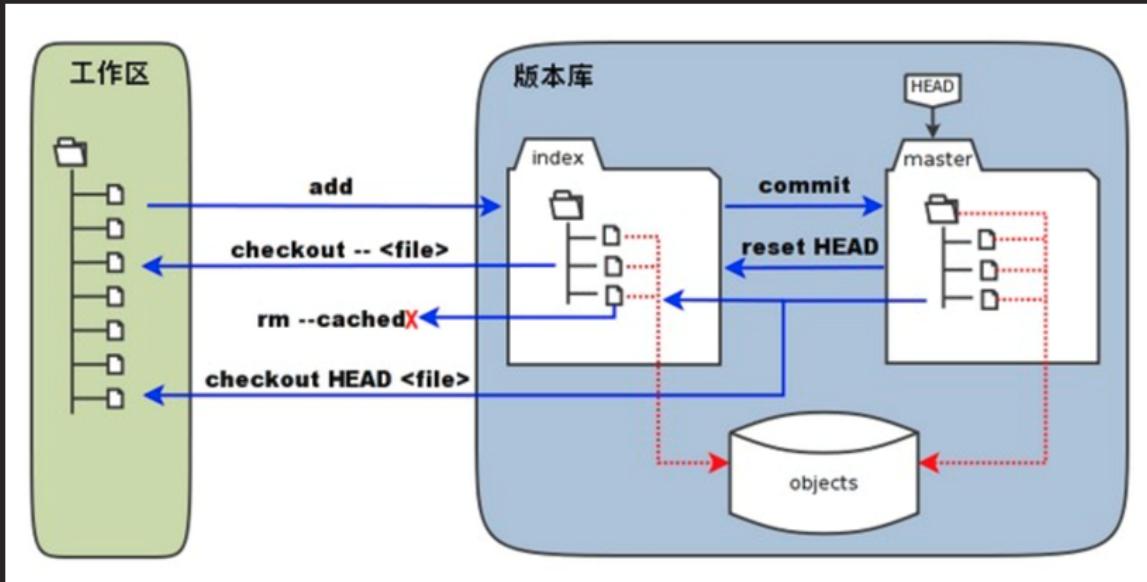
知识讲解

- **工作区**：就是你在电脑里能看到的目录
- **暂存区**：英文叫stage, 或index。一般存放在 ".git 目录下" 下的index文件 (.git/index) 中，所以我们把暂存区有时也叫作索引 (index)
- **版本库**：工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库



工作区、暂存区和版本库（续1）

知识讲解



创建仓库

知识讲解

- Git 使用 `git init` 命令来初始化一个 Git 仓库，Git 的很多命令都需要在 Git 的仓库中运行，所以 `git init` 是使用 Git 的第一个命令。

```
[root@localhost ~]# mkdir devops
[root@localhost ~]# cd devops/
[root@localhost devops]# git init
或
[root@localhost ~]# git init devops
```



添加文件到暂存区

知识讲解

- 添加指定文件

```
[root@localhost devops]# echo 'print("hello world!")' > hello.py  
[root@localhost devops]# git add hello.py  
[root@localhost devops]# git status
```

- 添加所有文件

```
[root@localhost devops]# cp hello.py welcome.py  
[root@localhost devops]# git add .  
[root@localhost devops]# git status -s
```



确认至仓库

知识讲解

- 提交之前务必先设置用户信息

```
[root@localhost devops]# git commit -m "初始化仓库"  
[root@localhost devops]# git status
```

- 添加追踪文件并提交到版本库

```
[root@localhost devops]# echo 'print("done.")' >> hello.py  
[root@localhost devops]# git commit -am "向hello.py添加新行"
```



删除跟踪文件

- 要从 Git 中移除某个文件，就必须要从已跟踪文件清单中移除，然后提交

```
[root@localhost devops]# git ls-files //查看版本库中文件  
[root@localhost devops]# git rm welcome.py  
[root@localhost devops]# git commit -m '删除welcome.py'
```

知识讲解



案例3：git本地操作

课堂练习

- 创建devops目录
- 为devops创建git仓库
- 新建文件hello.py，并将文件初始化到仓库中
- 修改hello.py并将其更新到仓库
- 从他库中删除hello.py



使用远程服务器

搭建本地gitlab服务器

知识讲解

- 导入中文版gitlab镜像

```
[root@localhost devops]# docker load < /path/to/gitlab_zh.tar
```

- 将物理主机ssh端口改为2022后，起动容器

```
[root@localhost devops]# docker run -d -h gitlab --name gitlab -p  
443:443 -p 80:80 -p 22:22 --restart always -v  
/srv/gitlab/config:/etc/gitlab -v /srv/gitlab/logs:/var/log/gitlab -v  
/srv/gitlab/data gitlab_zh:latest
```



初始化gitlab服务器

- 密码需大于8位



请为您的新帐户创建密码。

GitLab 中文社区版

用于代码协作的开源软件

细粒度访问控制管理 git 仓库以保证代码安全。 使用合并请求进行代码审查并加强团体合作。 每个项目均有自己的问题跟踪和维基页面。

修改密码

新密码

确认新密码

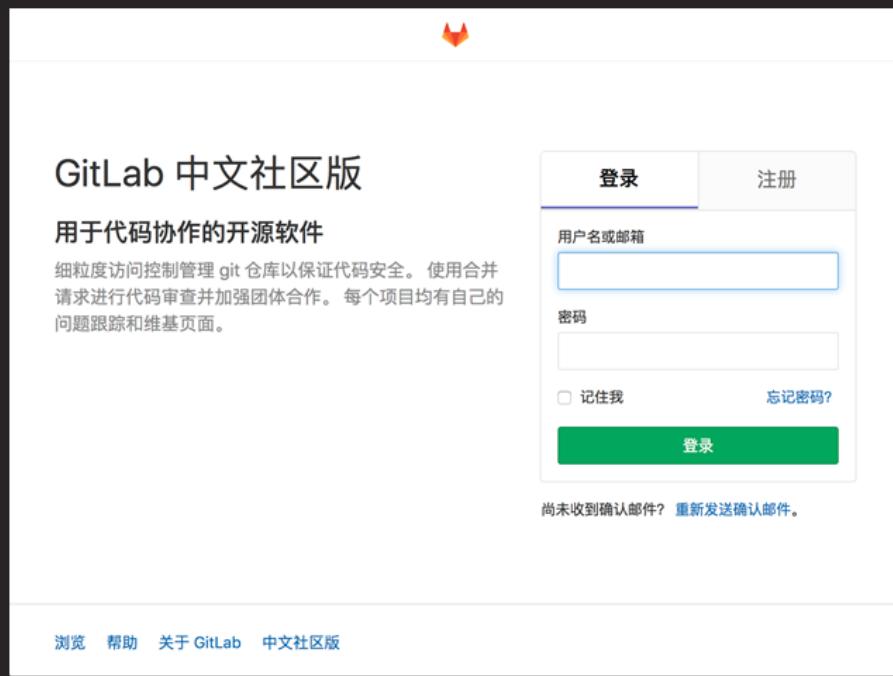
修改密码

没有收到确认邮件? 重新发送
已有账号和密码? 登录

浏览 帮助 关于 GitLab 中文社区版

初始化gitlab服务器 (续1)

- 默认用户名为root



GitLab 中文社区版

用于代码协作的开源软件

细粒度访问控制管理 git 仓库以保证代码安全。 使用合并请求进行代码审查并加强团体合作。 每个项目均有自己的问题跟踪和维基页面。

登录 注册

用户名或邮箱

密码

记住我 忘记密码?

登录

尚未收到确认邮件? 重新发送确认邮件。

浏览 帮助 关于 GitLab 中文社区版

添加gitlab项目

知识讲解

- 创建群组group
 - 使用群组管理项目和人员是非常好的方式
- 创建项目project
 - 存储代码的地方，里面还包含问题列表、维基文档以及其他一些Gitlab功能
- 创建成员member
 - 添加你的团队成员或其他人员到Gitlab



创建群组

知识讲解

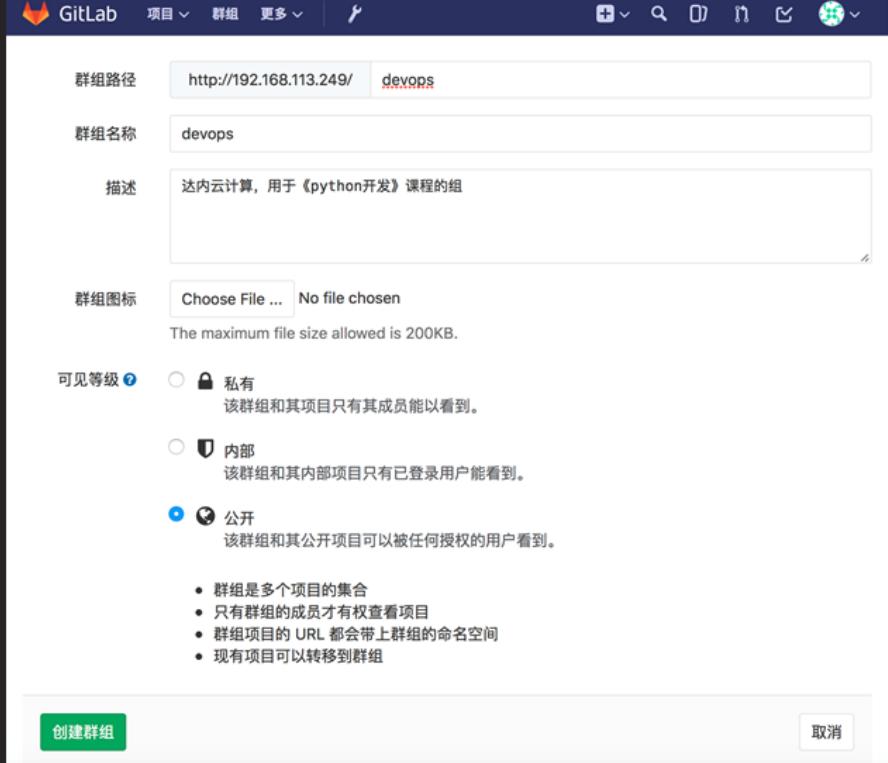
The screenshot shows the GitLab homepage with the following sections:

- 创建一个项目**: Projects are where you store your code, along with issue lists, wikis, and other GitLab features.
- 创建一个群组**: Groups are great for managing projects and members.
- 添加人员**: Add your team members or others to GitLab.
- 配置 GitLab**: Adjust your GitLab server settings.



创建群组（续1）

知识讲解



群组路径: http://192.168.113.249/ devops

群组名称: devops

描述: 达内云计算, 用于《python开发》课程的组

群组图标: Choose File ... No file chosen
The maximum file size allowed is 200KB.

可见等级: 私有
该群组和其项目只有其成员能以看到。
 内部
该群组和其内部项目只有已登录用户能看到。
 公开
该群组和其公开项目可以被任何授权的用户看到。

- 群组是多个项目的集合
- 只有群组的成员才有权查看项目
- 群组项目的 URL 都会带上群组的命名空间
- 现有项目可以转移到群组

创建群组 取消

知识讲解



创建项目

当前群组 搜索

群组 'devops' 创建成功。

devops

达内云计算, 用于《python开发》课程的组

按名称过滤... 最近创建 新项目

群组是几个项目的集合。
如果您在一个群组下组织项目, 它的工作方式就像一个文件夹。
您可以管理群组成员的权限并访问群组中的每个项目。

创建项目 (续1)

知识讲解



新建项目

项目可以用于存储你的文件（版本库）、安排你的工作（问题列表）、以及发布你的文档（维基页面）、以及一些其它事情。

项目建立后，所有这些功能都会被启用。不过你可以随后禁用那些你不需要的功能。

空白项目	从模板创建	导入项目
------	-------	------

项目路径: http://192.168.113.249/ devops 项目名称: core_py

希望将几个相关联的项目放置于同一个命名空间下? [创建群组](#)

项目描述 (可选): 用于存储核心语法规的代码

可见等级: 私有 (项目访问权限必须明确授权给每个用户。
 内部 (该项目允许已登录的用户访问。
 公开 (该项目允许任何人访问。)

[创建项目](#) [取消](#)

创建项目 (续2)

知识讲解



在账号中 [新建 SSH 公钥](#) 之前将无法通过 SSH 拉取或推送代码。 [不再显示 | 稍后提醒](#)

devops > core_py > 详细信息

项目 'core_py' 已创建成功。

core_py

用于存储核心语法规的代码

git@gitlab:devops/core_py.git

当前项目的版本仓库是空的

可以通过下面的命令行推送一个已存在的版本库。

或者可以从增加 README, a 许可证, or a .gitignore 文件开始本项目。

由于主分支(master)会被自动保护，只有当前项目的拥有者或者您具备主程序员权限才能进行初始推送。

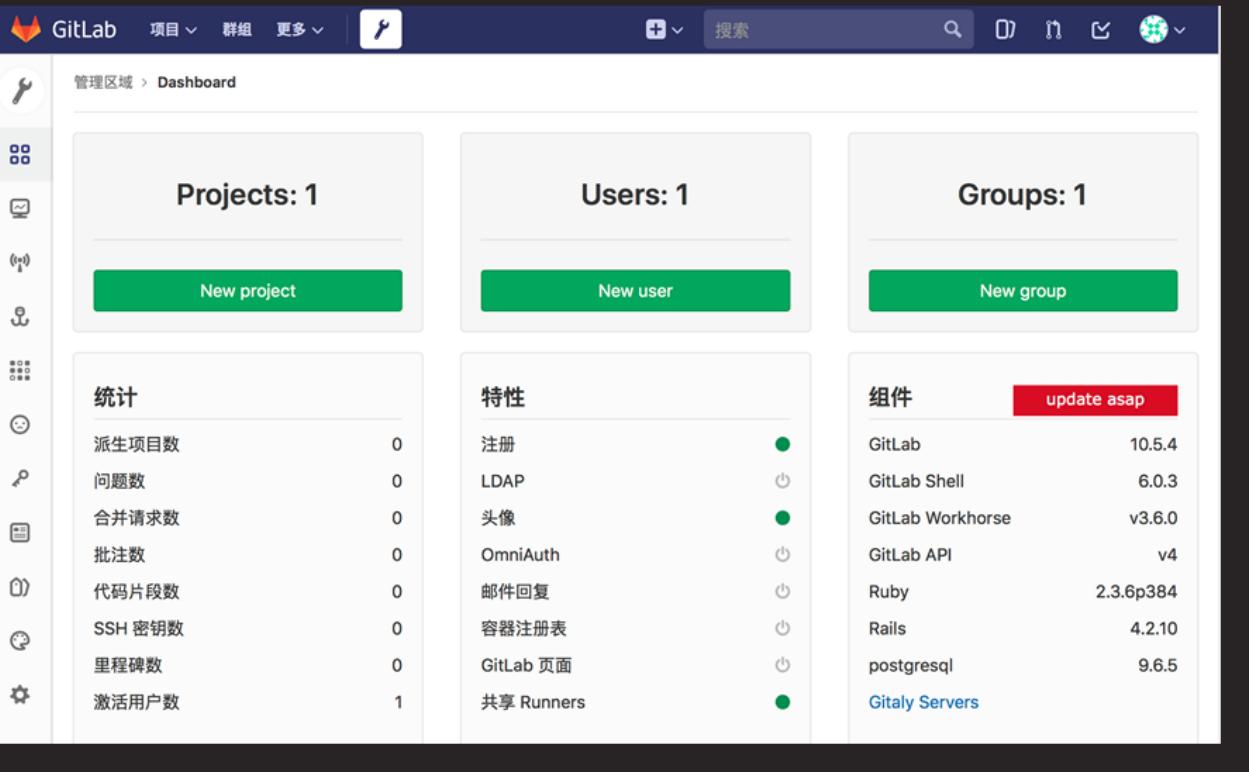
您可以为此项目激活 DevOps 自动化(测试版)。

将根据预定义的 CI/CD 配置自动构建、测试和部署应用程序。

[新建文件](#)

创建用户

知识讲解



管理区域 > Dashboard

Projects: 1	Users: 1	Groups: 1
New project	New user	New group

统计

派生项目数	0
问题数	0
合并请求数	0
批注数	0
代码片段数	0
SSH 密钥数	0
里程碑数	0
激活用户数	1

特性

注册	●
LDAP	○
头像	●
OmniAuth	○
邮件回复	○
容器注册表	○
GitLab 页面	○
共享 Runners	●

组件

组件	状态	版本
GitLab	●	10.5.4
GitLab Shell	○	6.0.3
GitLab Workhorse	●	v3.6.0
GitLab API	○	v4
Ruby	●	2.3.6p384
Rails	●	4.2.10
postgresql	○	9.6.5
Gitaly Servers	●	

创建用户 (续1)

- 创建用户后，再次编辑可设置密码

知识讲解



账号

姓名	张志刚
用户名	zhangzg
电子邮箱	zhangzg@tedu.cn

密码

密码 重置链接将自动生成临时密码发送给用户。
用户在第一次登录后需要强制修改密码。

权限

项目限制	100000
可以创建群组	<input checked="" type="checkbox"/>
权限级别	<input checked="" type="radio"/> 普通用户 普通用户可以访问他们的群组和项目
	<input type="radio"/> 管理员 管理员可以访问所有组，项目和用户，并且可以管理此安装中的所有功能

创建用户（续2）

- root用户将新用户加入组中，并设置新用户为“主程序员”



The screenshot shows the 'Members' page for the 'devops' group on GitLab. The interface includes a sidebar with navigation icons. The main area displays the 'Members' section with a search bar for '添加成员到 devops'. Two members are listed: 'Administrator @root' (加入时间 54 分钟前) and '张志刚 @zhangzg' (加入时间 33 分钟前). Both members are marked as '有权访问 devops 的成员' and have the '主程序员' role assigned.

创建用户（续3）

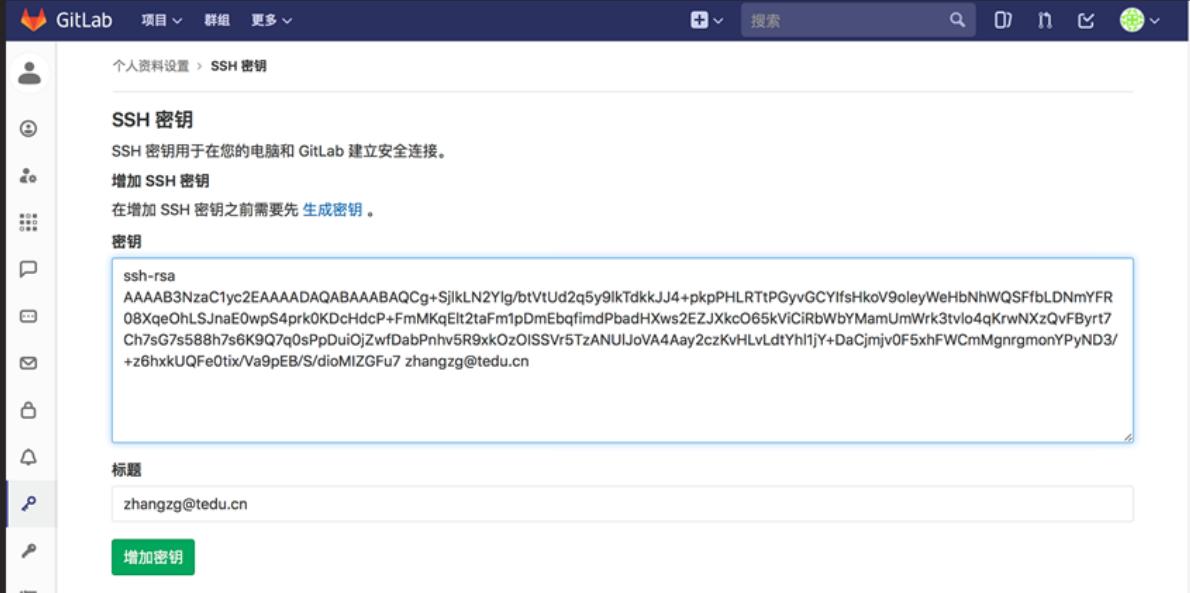
- 新用户初次登陆，需设置自己的密码



The screenshot shows the 'Set New Password' page on GitLab. The page title is '新密码 / 新密码' (New Password / New Password). It contains instructions: '请立即设置一个新密码。密码被成功修改后将会重定向到登录页面。' (Please set a new password immediately. After successful modification, you will be redirected to the login page.) Below this are three input fields: '当前密码' (Current Password), '密码' (Password), and '确认密码' (Confirm Password). A large green '设置新密码' (Set New Password) button is at the bottom.

用户管理

- 将本地生成的公钥上传至服务器



The screenshot shows the 'SSH 密钥' (SSH Keys) section of the GitLab user settings. It displays a long string of text representing an RSA public key. Below the key, there is a '标题' (Title) input field containing 'zhangzg@tedu.cn' and a green '增加密钥' (Add Key) button.

知识讲解



用户管理（续1）

- 将本地仓库推送至服务器

```
[root@localhost devops]# git remote rename origin old-origin
[root@localhost devops]# git remote add origin
git@192.168.113.249:/devops/core_py.git
[root@localhost devops]# git push -u origin --all
```

- 添加新文件

```
[root@localhost devops]# echo '# this is a test' > hi.py
[root@localhost devops]# git add hi.py
[root@localhost devops]# git commit hi.py -m '新的测试'
[root@localhost devops]# git push origin master
```

知识讲解



用户管理（续2）

知识讲解

- 下载代码到本地

```
[root@localhost ~]# git clone  
git@192.168.113.249:devops/core_py.git  
[root@localhost ~]# ls -a core_py/  
. . . .git hello.py hi.py
```

- 更新代码到本地

```
[root@localhost core_py]# git pull
```

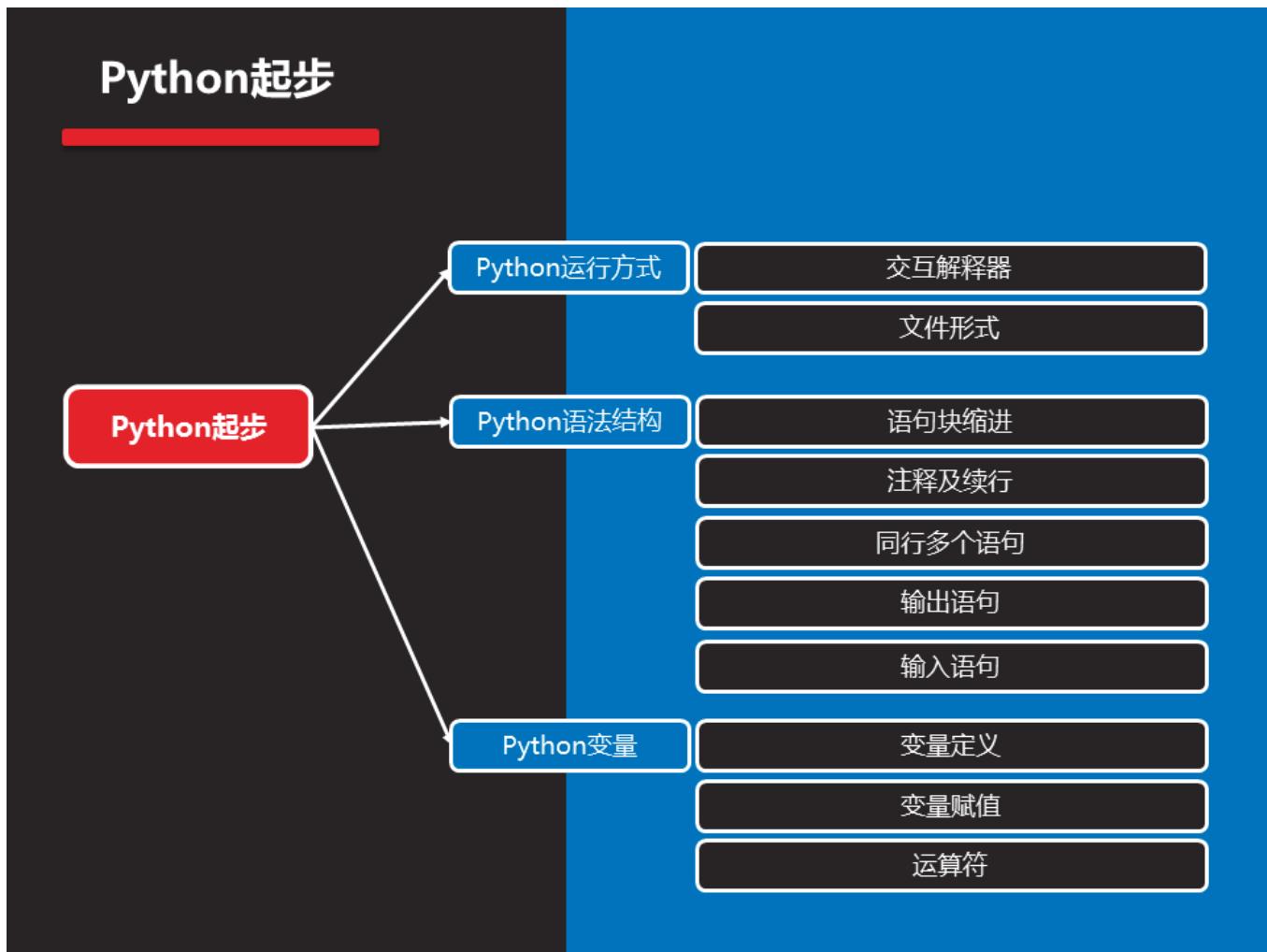


案例4：使用自建gitlab服务器

课堂练习

1. 通过docker搭建gitlab服务器
2. 新建群组devops
3. 新建项目core_py
4. 新建用户，他/她在devops组中是主程序员
5. 新用户上传版本库到gitlab
6. 熟悉git远程操作方法





Tedu.cn
达内教育

Python运行方式

交互解释器

知识讲解

- 进入交互解释器

```
[root@zzghost1 bin]# python3  
Python 3.6.3 (default, Oct 13 2017, 11:38:12)  
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

- 退出交互解释器

```
>>> exit()  
或  
>>> ctrl + d
```



文件形式

知识讲解

- 明确指定解释器

```
[root@zzghost1 day01]# python3 hello.py
```

- 赋予python文件可执行权限

```
[root@zzghost1 day01]# chmod +x hello.py  
[root@zzghost1 day01]# ./hello.py
```



Python语法结构

语句块缩进

- python代码块通过缩进对齐表达代码逻辑而不是使用大括号
- 缩进表达一个语句属于哪个代码块
- 缩进风格
 - 1或2：可能不够，很难确定代码语句属于哪个块
 - 8至10：可能太多，如果代码内嵌的层次太多，就会使得代码很难阅读
 - 4个空格：非常流行，范·罗萨姆支持的风格



语句块缩进（续1）

知识讲解

- 缩进相同的一组语句构成一个代码块，称之为代码组
- 首行以关键字开始，以冒号：结束，该行之后的一行或多行代码构成代码组
- 如果代码组只有一行，可以将其直接写在冒号后面，但是这样的写法可读性差，不推荐



注释及续行

知识讲解

- 首要说明的是：尽管Python是可读性最好的语言之一，这并不意味着程序员在代码中就可以不写注释
- 和很多UNIX脚本类似，python注释语句从#字符开始
- 注释可以在一行的任何地方开始，解释器会忽略掉该行#之后的所有内容
- 一行过长的语句可以使用反斜杠\分解成几行



同行多个语句

知识讲解

- 分号 ; 允许你将多个语句写在同一行上
- 但是些语句不能在这行开始一个新的代码块
- 因为可读会变差，所以不推荐使用



输出语句

知识讲解

- 获取帮助

```
>>> help(print)
```

- 使用方式

```
>>> print('Hello World!')  
>>> print('Hello' + 'World!')  
>>> print('Hello', 'World!')  
>>> print('Hello', 'World!', sep='***')  
>>> print('Hello', 'World!', sep='***', end='')
```



输入语句

- 获得帮助

```
>>> help(input)
```

- 使用方式（注意，返回值一定是字符类型）

```
>>> num = input("Number: ")
```

```
Number: 20
```

```
>>> num + 10
```

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

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

```
TypeError: must be str, not int
```

知识讲解



案例5：模拟用户登陆

1. 创建名为login.py的程序文件
2. 程序提示用户输入用户名
3. 用户输入用户名后，打印欢迎用户

课堂练习



Python变量

变量定义

- 变量名称约定
 - 第一个字符只能是大小写字母或下划线
 - 后续字符只能是大小写字母或数字或下划线
 - 区分大小写
- python是动态类型语言，即不需要预先声明变量的类型

知识讲解



变量定义（续1）

知识讲解

- 推荐采用的全名方法
 - 变量名全部采用小写字母
 - 简短、有意义
 - 多个单词间用下划线分隔
 - 变量名用名词，函数名用谓词（动词+名词）
 - 类名采用驼峰形式



变量赋值

知识讲解

- 变量的类型和值在赋值那一刻被初始化
- 变量赋值通过等号来执行

```
>>> counter = 0  
>>> name = 'bob'
```

- python也支持增量赋值

```
>>> n += 1    #等价于n = n + 1  
>>> n *= 1    #等价于n = n * 1  
>>> i++  
File "<stdin>", line 1  
    i++  
    ^
```

```
SyntaxError: invalid syntax
```



运算符

知识讲解

- 标准算术运算符
`+ - * / // % **`
- 比较运算符
`< <= > >= == != <>`
- 逻辑运算符
`and not or`



总结和答疑

起动容器失败

问题现象

故障分析及排除

总结和答疑

启动容器失败

问题现象

- 起动容器时，出现以下错误

Error response from daemon: driver failed programming external connectivity on endpoint gitlab
(58e237fda6825cb3fe5944425b09496417c0a17ba27f1ebff6bc49429
002a2a1): Error starting userland proxy: listen tcp 0.0.0.0:22: bind:
address already in use.

故障分析及排除

知识
讲解

- 原因分析
 - 在起动容器时，要将宿主机的22端口与容器的22端口进行映射
 - 提示地址已被占用
- 解决办法
 - 将宿主机ssh服务更换端口





Python开发入门

NSD PYTHON1

DAY02

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	数据类型概述
	10:30 ~ 11:20	
下午	11:30 ~ 12:00	判断语句
	14:00 ~ 14:50	
	15:00 ~ 15:50	while循环
	16:10 ~ 17:00	
	17:10 ~ 18:00	总结和答疑



数据类型概述

数据类型概述

数字

基本数字类型

数字表示方式

序列对象

定义字符串

字符串切片

字符串连接操作

定义列表

列表操作

元组的定义及操作

字典

字典的定义及操作

数据类型比较

数字

基本数字类型

- int : 有符号整数
- bool : 布尔值
 - True : 1
 - False : 0
- float : 浮点数
- complex : 复数

知识讲解



数字表示方式

知识讲解

- python默认以十进制数显示
- 数字以0o或0O开头表示为8进制数
- 数字以0x或0X开头表示16进制数
- 数字以0b或0B开头表示2进制数



字符串

定义字符串

知识讲解

- python中字符串被定义为引号之间的字符集合
- python支持使用成对的单引号或双引号
- 无论单引号，还是双引号，表示的意义相同
- python还支持三引号（三个连续的单引号或者双引号），可以用来包含特殊字符
- python不区分字符和字符串



字符串切片

知识讲解

- 使用索引运算符[]和切片运算符[:]可得到子字符串
- 第一个字符的索引是0，最后一个字符的索引是-1
- 子字符串包含切片中的起始下标，但不包含结束下标

```
>>> py_str = 'python'  
>>> py_str[0]  
'P'  
>>> py_str[-2]  
'o'  
>>> py_str[2:4]  
'th'  
>>> py_str[2:]  
'thon'  
>>> py_str[:4]  
'Pyth'
```



字符串连接操作

- 使用+号可以将多个字符串拼接在一起
- 使用*号可以将一个字符串重复多次

知识讲解

```
>>> py_str = 'python'  
>>> is_cool = 'is Cool'  
>>> print py_str + ' ' + is_cool  
python is Cool  
>>> py_str * 2  
'pythonpython'
```



定义列表

知识讲解

- 可以将列表当成普通的“数组”，它能保存任意数量任意类型的python对象
- 像字符串一样，列表也支持下标和切片操作
- 列表中的项目可以改变

```
>>> alist = [1, "tom", 2, "alice"]  
>>> alist[1] = 'bob'  
>>> alist[2:]
```



列表操作

- 使用in或not in判断成员关系
- 使用append方法向列表中追加元素

```
>>> alist = [1, "tom", 2, "alice"]
>>> 'tom' in alist
True
>>> 'alice' not in alist
False
>>> alist.append(3)
>>> alist[5] = 'bob'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

知识讲解



元组的定义及操作

- 可以认为元组是“静态”的列表
- 元组一旦定义，不能改变

```
>>> atuple = (1, "tom", 2, "alice")
>>> 'tom' in atuple
True
>>> atuple[0] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

知识讲解



字典

字典的定义及操作

- 字典是由键-值(key-value)对构成的映射数据类型
- 通过键取值，不支持下标操作

知识讲解

```
>>> user_dict = {'name':'bob', 'age':23}
>>> user_dict['gender'] = 'male'
>>> 'bob' in user_dict
False
>>> 'name' in user_dict
True
>>> user_dict[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```



数据类型比较

- 按存储模型分类
 - 标量类型：数值、字符串
 - 容器类型：列表、元组、字典
- 按更新模型分类：
 - 可变类型：列表、字典
 - 不可变类型：数字、字符串、元组
- 按访问模型分类
 - 直接访问：数字
 - 顺序访问：字符串、列表、元组
 - 映射访问：字典



判断语句

判断语句

if语句

if语句语法结构

if语句示例解析

扩展if语句

扩展if语句结构

扩展if语句示例解析

if语句

if语句语法结构

- 标准if条件语句的语法

```
if expression:  
    if_suite  
else:  
    else_suite
```

知识讲解

- 如果表达式的值非0或者为布尔值True, 则代码组if_suite被执行；否则就去执行else_suite
- 代码组是一个python术语，它由一条或多条语句组成，表示一个子代码块



if语句示例解析

- 只要表达式数字为非零值即为True

```
>>> if 10:  
...     print('Yes')  
Yes
```

- 空字符串、空列表、空元组，空字典的值均为False

```
>>> if "":  
...     print('Yes')  
... else:  
...     print('No')  
No
```

知识讲解



案例1：判断合法用户

课堂练习

- 创建login2.py文件
- 提示用户输入用户名和密码
- 获得到相关信息后，将其保存在变量中
- 如果用户输的用户名为bob，密码为123456，则输出Login successful，否则输出Login incorrect



扩展if语句

扩展if语句结构

- 扩展if语句结构

```
if expression1:  
    if_suite  
elif expression2:  
    elif_suite  
else:  
    else_suite
```

条件表达式

- 知识讲解
- Python 在很长的一段时间里没有条件表达式($C ? X : Y$)，或称三元运算符，因为范·罗萨姆一直拒绝加入这样的功能
 - 从Python 2.5集成的语法确定为： $X \text{ if } C \text{ else } Y$

```
>>> x, y = 3, 4  
>>> smaller = x if x < y else y  
>>> print smaller  
3
```



案例2：编写判断成绩的程序

- 课堂练习
- 创建grade.py脚本，根据用户输入的成绩分档，要求如下：
 1. 如果成绩大于60分，输出“及格”
 2. 如果成绩大于70分，输出“良”
 3. 如果成绩大于80分，输出“好”
 4. 如果成绩大于90分，输出“优秀”
 5. 否则输出“你要努力了”



案例3：编写石头剪刀布小游戏

课堂练习

- 编写game.py，要求如下：
 1. 计算机随机出拳
 2. 玩家自己决定如何出拳
 3. 代码尽量简化



while循环

while循环

循环语句基础

循环概述

while循环语法结构

循环语句进阶

break语句

continue语句

else语句

循环语句基础

循环概述

- 一组被重复执行的语句称之为循环体，能否继续重复，决定循环的终止条件
- Python中的循环有while循环和for循环
- 循环次数未知的情况下，建议采用while循环
- 循环次数可以预知的情况下，建议采用for循环

知识讲解



while循环语法结构

知识讲解

- 当需要语句不断的重复执行时，可以使用while循环

```
while expression:  
    while_suite
```

- 语句while_suite会被连续不断的循环执行，直到表达式的值变成0或False

```
sum100 = 0  
counter = 1
```

```
while counter <= 100:  
    sum100 += counter  
    counter += 1  
print ("result is %d" % sum100)
```



循环语句进阶

break语句

- break语句可以结束当前循环然后跳转到下条语句
- 写程序的时候，应尽量避免重复的代码，在这种情况下可以使用while-break结构

```
name = input('username: ')
while name != 'tom':
    name = input('username: ')
#可以替换为
while True:
    name = input('username: ')
    if name == 'tom':
        break
```

知识讲解



continue语句

- 当遇到continue语句时，程序会终止当前循环，并忽略剩余的语句，然后回到循环的顶端
- 如果仍然满足循环条件，循环体内语句继续执行，否则退出循环

```
sum100 = 0
counter = 0
while counter <= 100:
    counter += 1
    if counter % 2:
        continue
    sum100 += counter
print ("result is %d" % sum100)
```

知识讲解



else语句

- python中的while语句也支持else子句
- else子句只在循环完成后执行
- break语句也会跳过else块

```
sum10 = 0  
i = 1
```

```
while i <= 10:  
    sum10 += i  
    i += 1  
else:  
    print (sum10)
```

知识讲解



案例4：完善石头剪刀布小游戏

- 编写game2.py，要求如下：
 1. 基于上节game.py程序
 2. 实现循环结构，要求游戏三局两胜

课堂练习



案例5：猜数程序

课堂练习

- 编写guess.py，要求如下：
 1. 系统随机生成100以内的数字
 2. 要求用户猜生成的数字是多少
 3. 最多猜5次，猜对结束程序
 4. 如果5次全部猜错，则输出正确结果



总结和答疑



Python开发入门

NSD PYTHON1

DAY03

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	for循环
	10:30 ~ 11:20	
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	文件对象
	15:00 ~ 15:50	函数基础
	16:10 ~ 17:00	模块基础
	17:10 ~ 18:00	总结和答疑



for循环

for循环

for循环详解

for循环语法结构

range函数

列表解析

for循环详解

for循环语法结构

- python中的for接受可迭代对象（例如序列或迭代器）作为其参数，每次迭代其中一个元素

for iter_var in iterable:
 suite_to_repeat

知识讲解

- 与while循环一样，支持break、continue、else语句
- 一般情况下，循环次数未知采用while循环，循环次数已知，采用for循环



range函数

知识讲解

- for循环常与range函数一起使用
- range函数提供循环条件
- range函数的完整语法为：
`range(start, end, step =1)`



案例1：斐波那契数列

课堂练习

1. 斐波那契数列就是某一个数，总是前两个数之和，比如0, 1, 1, 2, 3, 5, 8
2. 使用for循环和range函数编写一个程序，计算有10个数字的斐波那契数列
3. 改进程序，要求用户输入一个数字，可以生成用户需要长度的斐波那契数列



案例2：九九乘法表

课堂练习

1. 创建mtable.py程序
2. 乘序运行后，可以在屏幕上打印出九九乘法表
3. 修改程序，由用户输入数字，可打印任意数字的乘法表

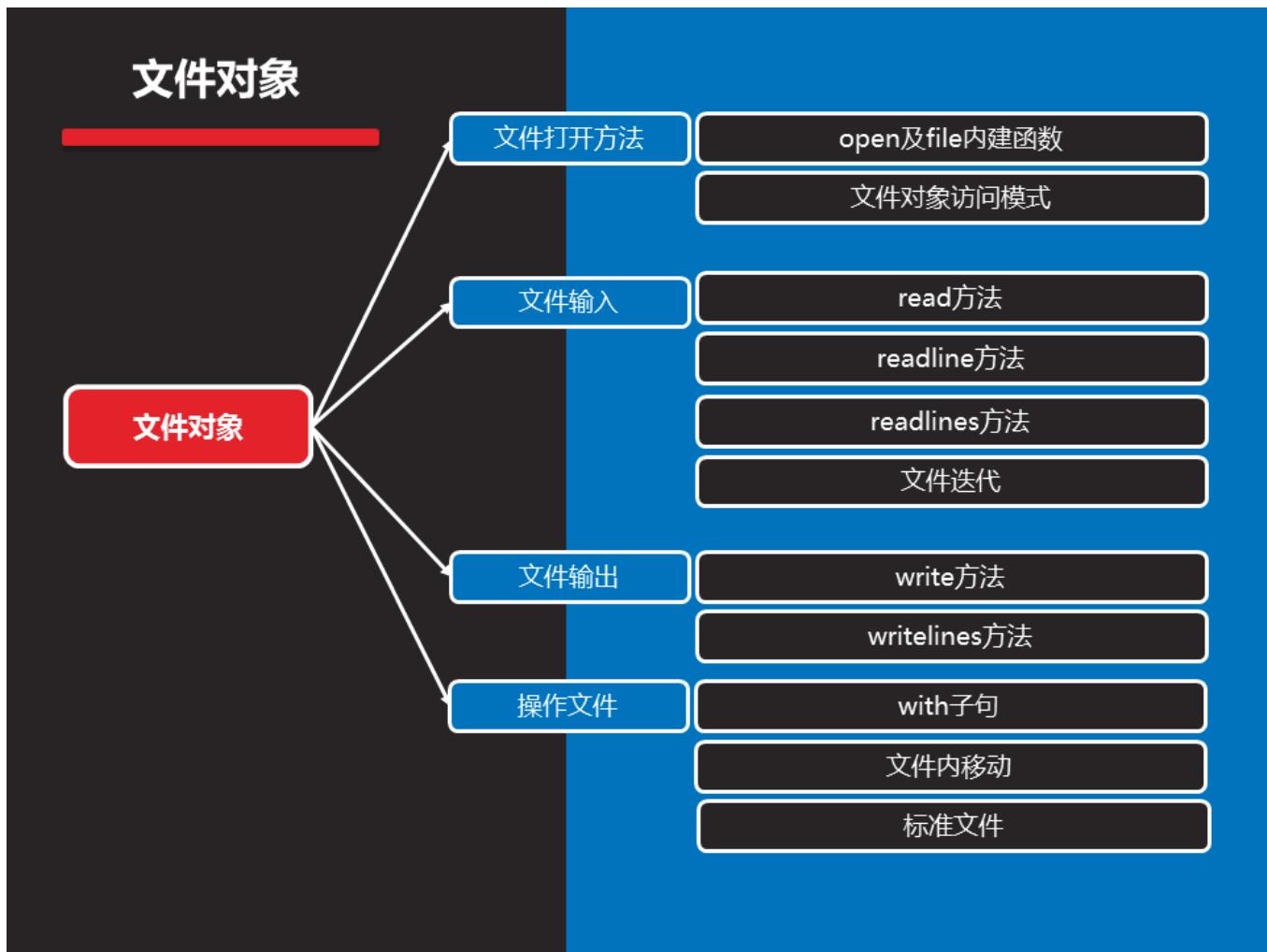


列表解析

知识讲解

- 它是一个非常有用、简单、而且灵活的工具，可以用来动态地创建列表
- 语法：
`[expr for iter_var in iterable]`
- 这个语句的核心是for循环，它迭代iterable对象的所有条目
- expr应用于序列的每个成员，最后的结果值是该表达式产生的列表





Tedu.cn
达内教育

文件打开方法

open及file内建函数

知识讲解

- 作为打开文件之门的“钥匙”，内建函数open()以及file()提供了初始化输入/输出(I/O)操作的通用接口
- 成功打开文件后时候会返回一个文件对象，否则引发一个错误
- open()方法和file()方法可以完全相互替换
- 基本语法：

```
file_object = open(file_name, access_mode='r', buffering=-1)
```



文件对象访问模式

知识讲解

文件模式	操作
r	以读方式打开(文件不存在则报错)
w	以写方式打开(文件存在则清空, 不存在则创建)
a	以追加模式打开(必要时创建新文件)
r+	以读写模式打开(参见r)
w+	以读写模式打开(参见w)
a+	以读写模式打开(参见a)
b	以二进制模式打开



文件输入

read方法

知识讲解

- `read()`方法用来直接读取字节到字符串中，最多读取给定数目个字节
- 如果没有给定size参数（默认值为-1）或者size值为负，文件将被读取直至末尾

```
>>> data = fobj.read()  
>>> print(data)
```



readline方法

- 读取打开文件的一行（读取下个行结束符之前的所有字节）
- 然后整行，包括行结束符，作为字符串返回
- 它也有一个可选的size参数，默认为-1，代表读至行结束符
- 如果提供了该参数，那么在超过size个字节后会返回不完整的行

```
>>> data = fobj.readline()  
>>> print(data)
```



readlines方法

- readlines()方法读取所有（剩余的）行然后把它们作为一个字符串列表返回

```
>>> data = fobj.readlines()  
>>> print(data)
```



文件迭代

- 如果需要逐行处理文件，可以结合for循环迭代文件
- 迭代文件的方法与处理其他序列类型的数据类似

```
>>> fobj = open('star.py')  
>>> for eachLine in fobj:  
...     print(eachLine, end= '')
```

知识讲解



文件输出

write方法

知识讲解

- write()内建方法功能与read()和readline()相反。它把含有文本数据或二进制数据块的字符串写入到文件中去
- 写入文件时，不会自动添加行结束标志，需要程序员手工输入

```
>>> fobj.write('Hello World!\n')
```

13



writelines方法

知识讲解

- 和readlines()一样，writelines()方法是针对列表的操作
- 它接受一个字符串列表作为参数，将它们写入文件
- 行结束符并不会被自动加入，所以如果需要的话，必须在调用writelines()前给每行结尾加上行结束符

```
>>> fobj.writelines(['Hello World!\n', 'python programing\n'])
```



操作文件

with子句

- with语句是用来简化代码的
- 在将打开文件的操作放在with语句中，代码块结束后，文件将自动关闭

```
>>> with open('foo.py') as f:  
...     data = f.readlines()  
...  
>>> f.closed  
True
```

文件内移动

知识讲解

- `seek(offset[, whence])` : 移动文件指针到不同的位置
 - offset是相对于某个位置的偏移量
 - whence的值，0表示文件开头，1表示当前位置，2表示文件的结尾
- `tell()` : 返回当前文件指针的位置



标准文件

知识讲解

- 程序一执行，就可以访问三个标准文件
 - 标准输入：一般是键盘，使用`sys.stdin`
 - 标准输出：一般是显示器缓冲输出，使用`sys.stdout`
 - 标准错误：一般是显示器的非缓冲输出，使用`sys.stderr`

```
>>> import sys  
>>> sys.stdout.write('hello world!\n')  
hello world!  
>>> hi = sys.stdin.readline()  
hello  
>>> hi  
'hello\n'
```



案例3：模拟cp操作

课堂练习

1. 创建cp.py文件
2. 将/bin/ls “拷贝”到/root/目录下
3. 不要修改原始文件



函数基础

函数基础

函数基本操作

函数基本概念

创建函数

调用函数

函数的返回值

函数参数

定义参数

传递参数

位置参数

默认参数

函数基本操作

函数基本概念

- 函数是对程序逻辑进行结构化或过程化的一种编程方法
- 将整块代码巧妙地隔离成易于管理的小块
- 把重复代码放到函数中而不是进行大量的拷贝，这样既能节省空间，也有助于保持一致性
- 通常函数都是用于实现某一种功能

知识讲解



创建函数

知识讲解

- 函数是用def语句来创建的，语法如下：

```
def function_name(arguments):
    "function_documentation_string"
    function_body_suite
```

- 标题行由def关键字，函数的名字，以及参数的集合（如果有的话）组成
- def子句的剩余部分包括了一个虽然可选但是强烈推荐的文档字符串，和必需的函数体



调用函数

知识讲解

- 同大多数语言相同，python用一对圆括号调用函数
- 如果没有加圆括号，只是对函数的引用

```
>>> def foo():
...     print('hello')
...
>>> foo()
hello
>>> foo
<function foo at 0x7ff2328967d0>
```



函数的返回值

知识讲解

- 多数情况下，函数并不直接输出数据，而是向调用者返回值
- 函数的返回值使用return关键字
- 没有return的话，函数默认返回None

```
>>> def foo():
...     res = 3 + 4
>>> i = foo()
>>> print i
None
```



函数参数

定义参数

知识讲解

- 形式参数
 - 函数定义时，紧跟在函数名后（圆括号内）的参数被称为形式参数，简称形参。由于它不是实际存在变量，所以又称虚拟变量
- 实际参数
 - 在主调函数中调用一个函数时，函数名后面括弧中的参数（可以是一个表达式）称为“实际参数”，简称实参



传递参数

知识讲解

- 调用函数时，实参的个数需要与形参数一致
- 实参将依次传递给形参

```
>>> def foo(x, y):  
...     print('x=%d, y=%d' % (x, y))  
>>> foo()  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: foo() takes exactly 2 arguments (0 given)  
>>> foo(3)  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: foo() takes exactly 2 arguments (1 given)  
>>> foo(3, 4)  
x=3, y=4
```



位置参数

- 与shell脚本类似，程序名以及参数都以位置参数的方式传递给python程序
- 使用sys模块的argv列表接收

```
[root@zzghost1 day02]# vim args.py
#!/usr/bin/env python3
import sys
print(sys.argv)
```

```
[root@zzghost1 day02]# ./args.py hello world
['./args.py', 'hello', 'world']
```

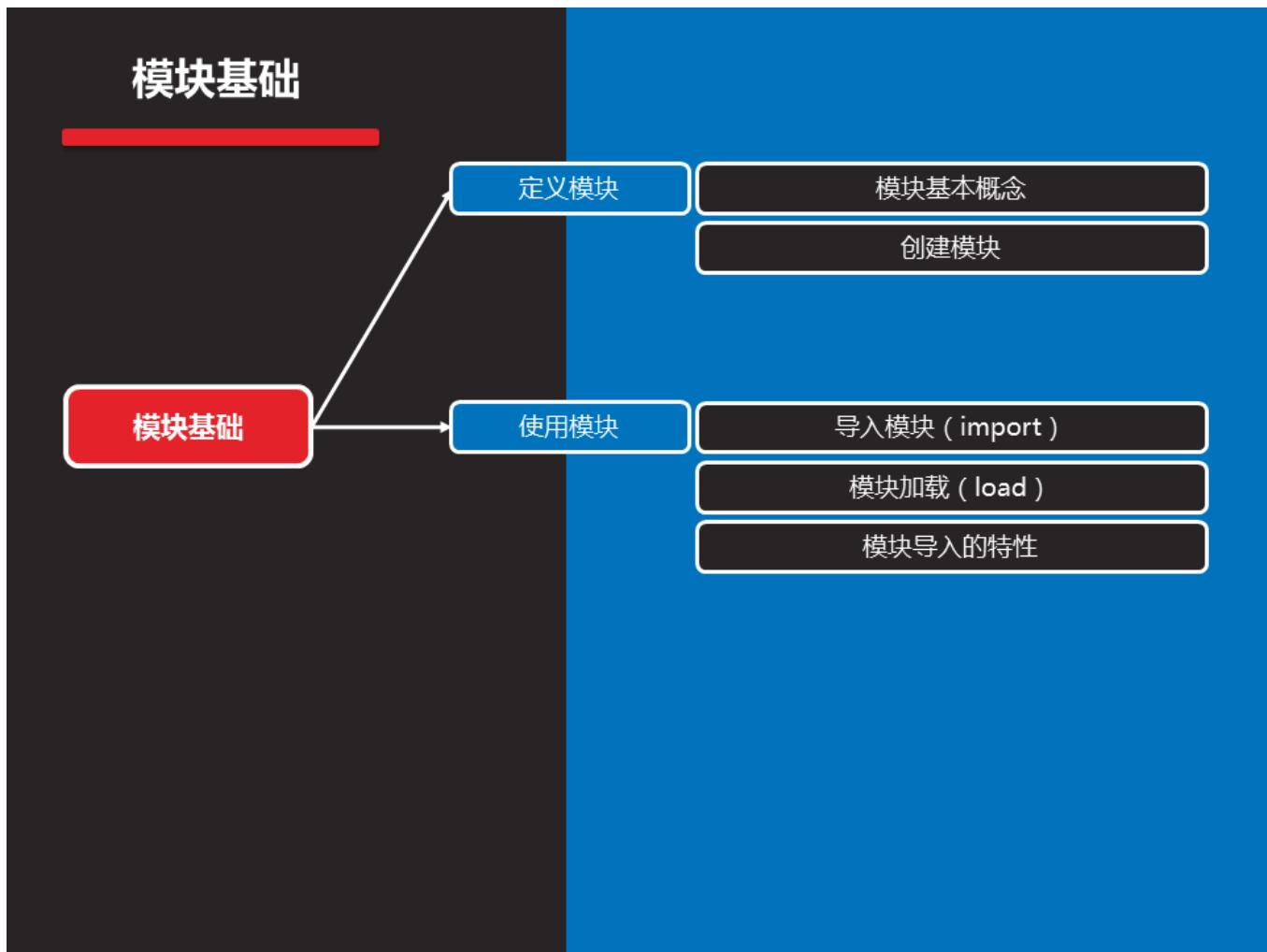


默认参数

- 默认参数就是声明了默认值的参数
- 因为给参数赋予了默认值，所以在函数调用时，不向该参数传入值也是允许的

```
>>> def pstar(num = 30):
...     print('*' * num)
...
>>> pstar()
*****
>>> pstar(40)
*****
```





定义模块

模块基本概念

知识讲解

- 模块是从逻辑上组织python代码的形式
- 当代码量变得相当大的时候，最好把代码分成一些有组织的代码段，前提是保证它们的彼此交互
- 这些代码片段相互间有一定的联系，可能是一个包含数据成员和方法的类，也可能是一组相关但彼此独立的操作函数



创建模块

知识讲解

- 模块物理层面上组织模块的方法是文件，每一个以.py作为结尾的python文件都是一个模块
- 模块名称切记不要与系统中已存在的模块重名
- 模块文件名字去掉后面的扩展名（.py）即为模块名



导入模块 (import)

知识讲解

- 使用import导入模块
- 模块被导入后，程序会自动生成pyc的字节码文件以提升性能
- 模块属性通过“模块名.属性”的方法调用
- 如果仅需要模块中的某些属性，也可以单独导入

```
>>> import sys  
>>> import os, string  
>>> string.digits  
'0123456789'  
>>> from random import randint  
>>> randint(1, 10)  
3
```



模块加载 (load)

知识讲解

- 一个模块只被加载一次，无论它被导入多少次
- 只加载一次可以阻止多重导入时代码被多次执行
- 如果两个文件相互导入，防止了无限的相互加载
- 模块加载时，顶层代码会自动执行，所以只将函数放入模块的顶层是良好的编程习惯



模块导入的特性

知识讲解

- 模块具有一个`_name_`特殊属性
- 当模块文件直接执行时，`_name_`的值为'`_main_`'
- 当模块被另一个文件导入时，`_name_`的值就是该模块的名字

```
[root@zzghost1 day02]# vim foo.py
#!/usr/bin/env python3
print(__name__)
[root@py01 bin]# ./foo.py
__main__
[root@zzghost1 day02]# python
>>> import foo
foo
```



案例4：生成随机密码

课堂练习

- 创建`randpass.py`脚本，要求如下：
 1. 编写一个能生成8位随机密码的程序
 2. 使用`random`的`choice`函数随机取出字符
 3. 改进程序，用户可以自己决定生成多少位的密码



总结和答疑



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	
下午	11:30 ~ 12:00	字符串详解
	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

shell相关模块

shutil模块

复制和移动

目录操作

权限管理

语法风格

变量赋值

合法标识符

关键字

内建

模块结构及布局

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)

知识讲解

- `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

知识讲解

- `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

知识讲解

- `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`

知识讲解



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

知识讲解



关键字

知识讲解

- 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

知识讲解

- 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去

```
#!/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()
```

知识讲解

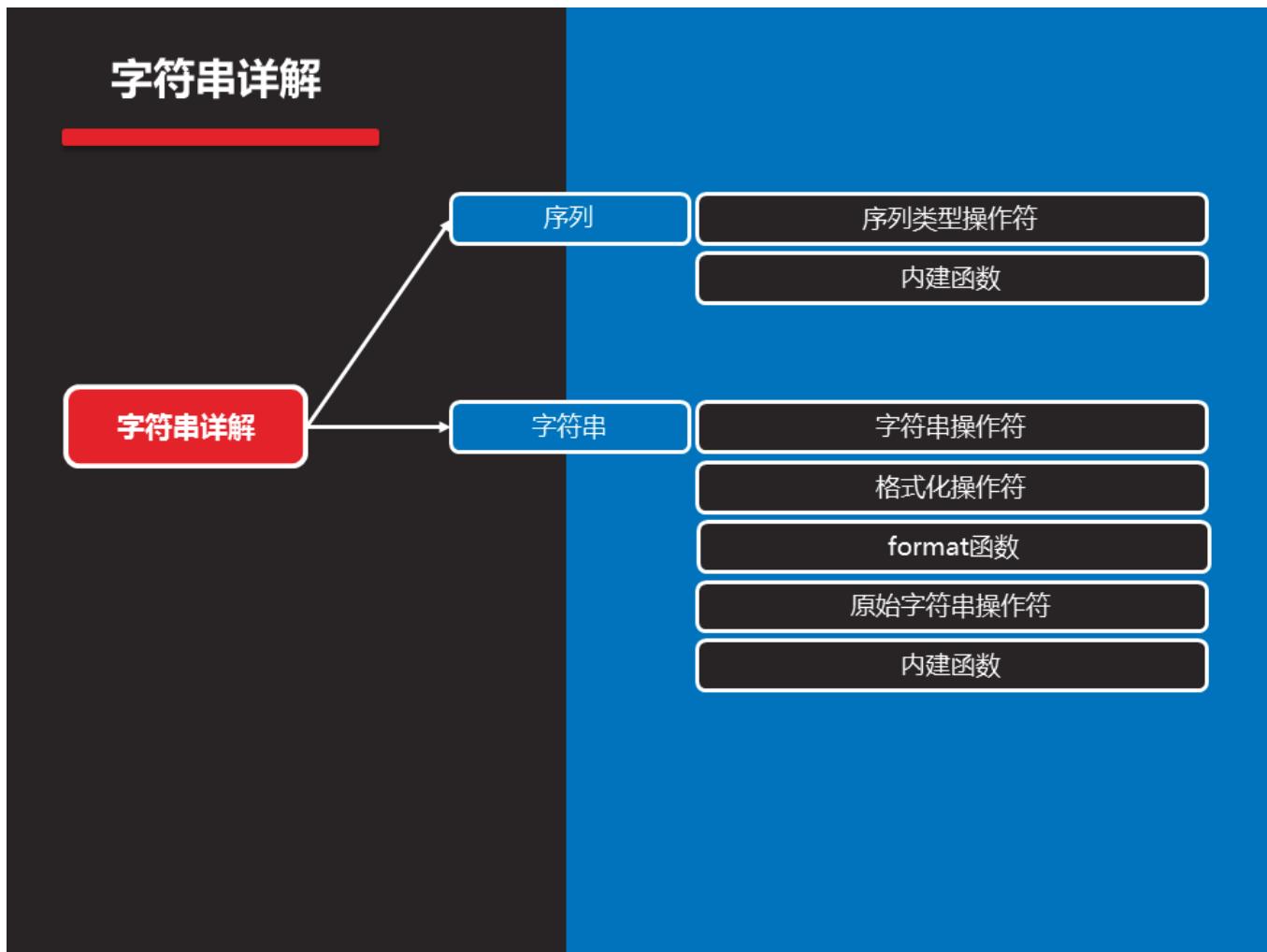


案例1：创建文件

课堂练习

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





Tedu.cn
达内教育

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）

知识讲解

- 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）

知识讲解

- 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

课堂练习

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

知识讲解

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

知识讲解



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])

知识讲解



案例3：创建用户

课堂练习

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

知识讲解

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



案例4：格式化输出

课堂练习

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

知识讲解

- `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）

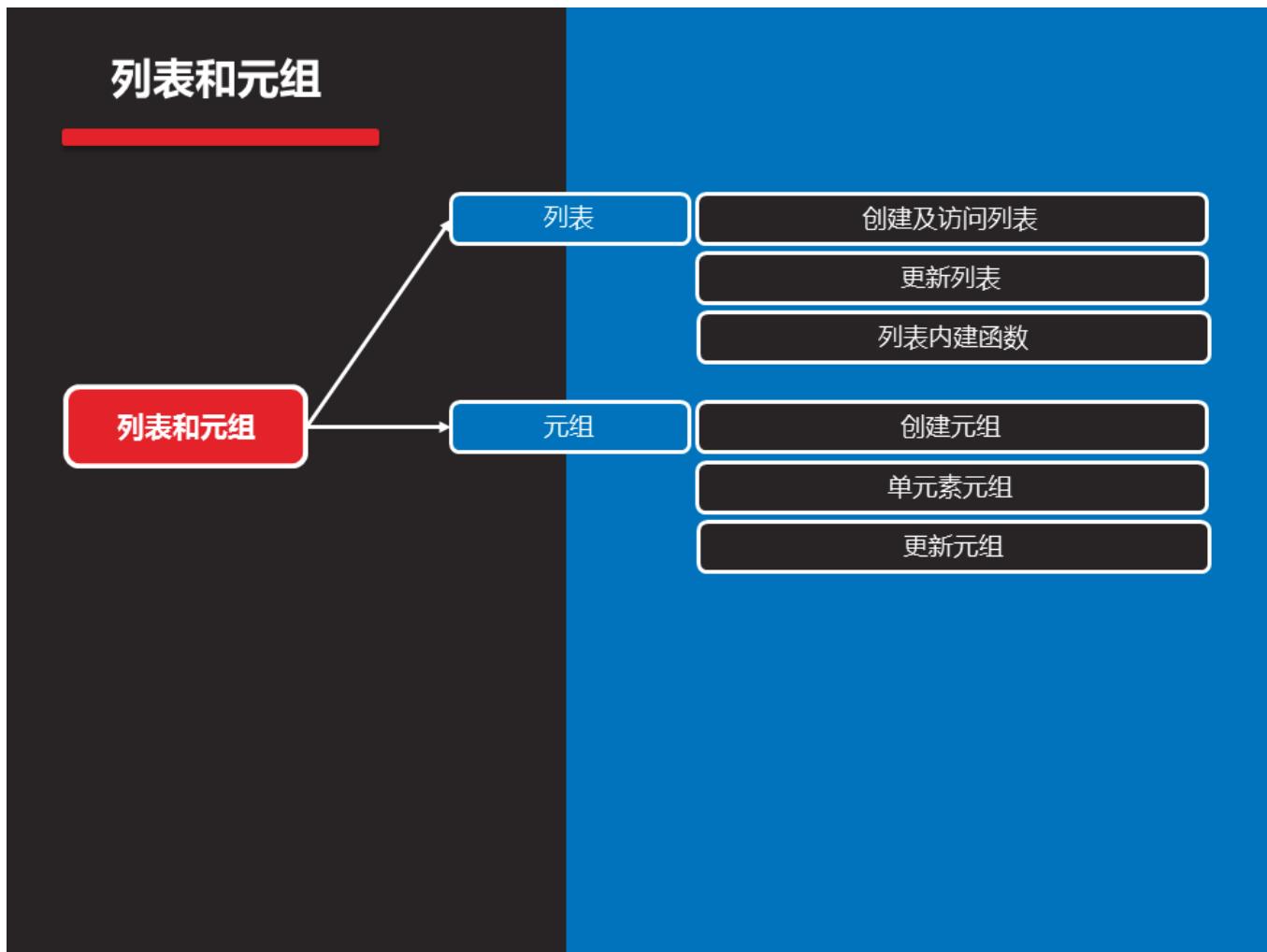
- `string.endswith(obj, beg=0,end=len(string))`：检查字符串是否以obj结束，如果beg或者end指定则检查指定的范围内是否以obj结束，如果是，返回True，否则返回False
- `string.islower()`：如果string中包含至少一个区分大小写的字符，并且所有这些字符都是小写，则返回True，否则返回False



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

创建及访问列表

知识讲解

- 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

知识讲解

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



元组

创建元组

知识讲解

- 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

知识讲解

- 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])
```

知识讲解



案例5：用列表构建栈结构

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

课堂练习



总结和答疑