Table of Contents

软件测试**Python**课程

本阶段课程不仅可以帮助我们进入Python语言世界，同时也是后续UI自动化测试、接口自动化测试等课程阶段的语言基础。

> Life is short, you need Python! -- 人生苦短，我用Python！

课程大纲

序号	章节	知识点
1	Python基础	1. 认识Python 2. Python环境搭建 3. PyCharm 4. 注释、变量、变量类型、输入输出、运算符
2	流程控制结构	1. 判断语句 2. 循环
3	数据序列	1. 字符串 2. 列表 3. 元组 4. 字典
4	函数	1. 函数基础 2. 变量进阶 3. 函数进阶 4. 匿名函数
5	面向对象	1. 面向对象编程介绍 2. 类和对象 3. 面向对象基础语法 4. 封装、继承、多态 5. 类属性和类方法
6	异常、模块、文件操作	1. 异常 2. 模块和包 3. 文件操作
7	UnitTest框架	1. UnitTest基本使用 2. UnitTest断言 3. 参数化 4. 生成HTML测试报告

课程目标

1. 掌握如何搭建Python开发环境；
2. 掌握Python基础语法, 具备基础的编程能力；
3. 建立编程思维以及面向对象程序设计思想；
4. 掌握如何通过UnitTest编写测试脚本，并生成HTML测试报告。

异常、模块、文件操作

目标

1. 掌握Python中异常的概念
2. 掌握Python中处理异常的结构
3. 掌握Python中抛出异常的结构
4. 掌握Python中模块的含义
5. 掌握Python中完成模块及包的导入、使用
6. 掌握文件的概念和作用
7. 掌握文件的基本操作
8. 掌握JSON的语法格式
9. 熟练掌握对JSON数据的操作

异常

目标

1. 理解异常的概念
2. 掌握捕获异常
3. 掌握异常的传递
4. 掌握抛出异常

1. 异常的概念

- 程序在运行时，如果 `Python 解释器` 遇到到一个错误，会停止程序的执行，并且提示一些错误信息，这就是异常
- 程序停止执行并且提示错误信息 这个动作，我们通常称之为：抛出**(raise)**异常

> 程序开发时，很难将 所有的特殊情况 都处理的面面俱到，通过 异常捕获 可以针对突发事件做集中的处理，从而保证程序的 稳定性和健壮性

2. 捕获异常

2.1 简单的捕获异常语法

在程序开发中，如果 对某些代码的执行不能确定是否正确，可以增加 `try(尝试)` 来 捕获异常

捕获异常最简单的语法格式：

```
try:
    尝试执行的代码
except:
    出现错误的处理
```

- `try` 尝试，下方编写要尝试代码，不确定是否能够正常执行的代码
- `except` 如果有异常捕获，下方编写捕获到异常,处理失败的代码

简单异常捕获演练-要求用户输入整数

```
try:
    # 提示用户输入一个数字
    num = int(input("请输入数字：："))
except:
    print("请输入正确的数字")
```

2.2 错误类型捕获

- 在程序执行时，可能会遇到 不同类型的异常，并且需要 针对不同类型的异常，做出不同的响应，这个时候，就需要捕获错误类型了
- 语法如下：

```
try:
  # 尝试执行的代码
  pass
except 错误类型1:
  # 针对错误类型1，对应的代码处理
  pass
except (错误类型2, 错误类型3):
  # 针对错误类型2 和 3，对应的代码处理
  pass
except Exception as result:
  print("未知错误 %s" % result)
```

- 当 `Python` 解释器 抛出异常 时，最后一行错误信息的第一个单词，就是错误类型

异常类型捕获演练-要求用户输入整数

需求

1. 提示用户输入一个整数
2. 使用 `8` 除以用户输入的整数并且输出

```
try:
    num = int(input("请输入整数："))
    result = 8 / num
    print(result)
except ValueError:
    print("请输入正确的整数")
except ZeroDivisionError:
    print("除 0 错误")
```

捕获未知错误

- 在开发时，要预判到所有可能出现的错误，还是有一定难度的
- 如果希望程序 无论出现任何错误，都不会因为 `Python` 解释器 抛出异常而被终止，可以再增加一个 `except Exception`
- exception Exception as result: 中 result记录异常的错误信息

语法如下：

```
except Exception as result:
    print("未知错误 %s" % result)
```

2.3 异常捕获完整语法

- 在实际开发中，为了能够处理复杂的异常情况，完整的异常语法如下：

> 提示：
>
> - 有关完整语法的应用场景，在后续学习中，结合实际的案例会更好理解
> - 现在先对这个语法结构有个印象即可

```
try:
    # 尝试执行的代码
    pass
except 错误类型1:
    # 针对错误类型1，对应的代码处理
```

```
    pass
except 错误类型2:
    # 针对错误类型2，对应的代码处理
    pass
except (错误类型3, 错误类型4):
    # 针对错误类型3 和 4，对应的代码处理
    pass
except Exception as result:
    # 打印错误信息
    print(result)
else:
    # 没有异常才会执行的代码
    pass
finally:
    # 无论是否有异常，都会执行的代码
    print("无论是否有异常，都会执行的代码")
```

- `else` 只有在没有异常时才会执行的代码
- `finally` 无论是否有异常，都会执行的代码

- 之前一个演练的 完整捕获异常 的代码如下：

```
try:
    num = int(input("请输入整数："))
    result = 8 / num
    print(result)
except ValueError:
    print("请输入正确的整数")
except ZeroDivisionError:
    print("除 0 错误")
except Exception as result:
    print("未知错误 %s" % result)
else:
    print("正常执行")
finally:
    print("执行完成，但是不保证正确")
```

3. 异常的传递

- 异常的传递 —— 当 函数/方法 执行 出现异常，会 将异常传递 给 函数/方法 的 调用一方
- 如果 传递到主程序，仍然 没有异常处理，程序才会被终止

> 提示

- 在开发中，可以在主函数中增加 异常捕获
- 而在主函数中调用的其他函数，只要出现异常，都会传递到主函数的 异常捕获 中
- 这样就不需要在代码中，增加大量的 异常捕获，能够保证代码的整洁

需求

1. 定义函数 `demo1()` 提示用户输入一个整数并且返回
2. 定义函数 `demo2()` 调用 `demo1()`
3. 在主程序中调用 `demo2()`

```
def demo1():
    return int(input("请输入一个整数："))


def demo2():
```

```
    return demo1()

try:
    print(demo2())
except ValueError:
    print("请输入正确的整数")
except Exception as result:
    print("未知错误 %s" % result)
```

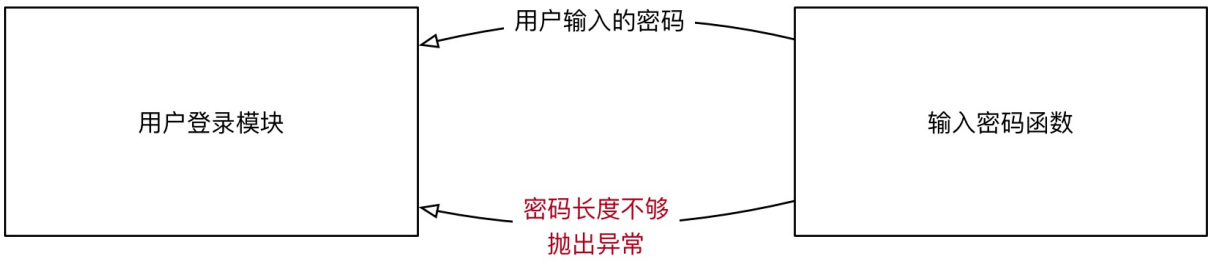
4. 抛出 `raise` 异常

4.1 应用场景

- 在开发中，除了 代码执行出错 `Python` 解释器会 抛出 异常之外
- 还可以根据 应用程序 特有的业务需求 主动抛出异常

示例

- 提示用户 输入密码，如果 长度少于 **8**，抛出 异常



注意

- 当前函数 只负责 提示用户输入密码，如果 密码长度不正确，需要其他的函数进行额外处理
- 因此可以 抛出异常，由其他需要处理的函数 捕获异常

4.2 抛出异常

- `Python` 中提供了一个 `Exception` 异常类
- 在开发时，如果满足 特定业务需求时，希望 抛出异常，可以：
  1. 创建 一个 `Exception类` 的 对象
  2. 使用 `raise` 关键字 抛出 异常对象

需求

- 定义 `input_password` 函数，提示用户输入密码
- 如果用户输入长度 < 8，抛出异常
- 如果用户输入长度 >=8，返回输入的密码

```
def input_password():
    # 1. 提示用户输入密码
    pwd = input("请输入密码：")

    # 2. 判断密码长度，如果长度 >= 8，返回用户输入的密码
    if len(pwd) >= 8:
        return pwd

    # 3. 密码长度不够，需要抛出异常
```

```
    # 1> 创建异常对象 - 使用异常的错误信息字符串作为参数
    ex = Exception("密码长度不够")

    # 2> 抛出异常对象
    raise ex


try:
    user_pwd = input_password()
    print(user_pwd)
except Exception as result:
    print("发现错误: %s" % result)
```

模块和包

目标

```
1. 掌握模块的概念和导入
2. 掌握包的创建和导入
```

1. 模块

1.1 模块的概念

> 模块是 **Python** 程序架构的一个核心概念

- 每一个以扩展名 `py` 结尾的 `Python` 源代码文件都是一个 模块
- 模块名 同样也是一个 标识符，需要符合标识符的命名规则
- 在模块中定义的 全局变量 、函数、类 都是提供给外界直接使用的 工具
- 模块 就好比是 工具包，要想使用这个工具包中的工具，就需要先 导入 这个模块

1.2 模块的两种导入方式

1）import 导入

```
import 模块名1, 模块名2
```

> 提示：在导入模块时，每个导入应该独占一行

```
import 模块名1
import 模块名2
```

- 导入之后
  - 通过 `模块名.` 使用 模块提供的工具 —— 全局变量、函数、类

使用 `as` 指定模块的别名

> 如果模块的名字太长，可以使用 `as` 指定模块的名称，以方便在代码中的使用

```
import 模块名1 as 模块别名
```

> 注意：模块别名 应该符合 大驼峰命名法

2）from...import 导入

- 如果希望 从某一个模块 中，导入 部分 工具，就可以使用 `from ... import` 的方式
- `import 模块名` 是 一次性 把模块中 所有工具全部导入，并且通过 模块名/别名 访问

```
# 从 模块 导入 某一个工具
```

```
from 模块名1 import 工具名
```

- 导入之后
  - 不需要 通过 `模块名.`
  - 可以直接使用 模块提供的工具 —— 全局变量、函数、类

注意

> 如果 两个模块，存在 同名的函数，那么 后导入模块的函数，会 覆盖掉先导入的函数

- 开发时 `import` 代码应该统一写在 代码的顶部，更容易及时发现冲突
- 一旦发现冲突，可以使用 `as` 关键字 给其中一个工具起一个别名

from...import *（知道）

```
# 从 模块 导入 所有工具
from 模块名1 import *
```

注意

> 这种方式不推荐使用，因为函数重名并没有任何的提示，出现问题不好排查

1.3 模块的搜索顺序[扩展]

`Python` 的解释器在 导入模块 时，会：

1. 搜索 当前目录 指定模块名的文件，如果有就直接导入
2. 如果没有，再搜索 系统目录

> 在开发时，给文件起名，不要和 系统的模块文件 重名

`Python` 中每一个模块都有一个内置属性 `__file__` 可以 查看模块 的 完整路径

示例

```
import random

# 生成一个 0~10 的数字
rand = random.randint(0, 10)

print(rand)
```

> 注意：如果当前目录下，存在一个 `random.py` 的文件，程序就无法正常执行了！

- 这个时候， `Python` 的解释器会 加载当前目录 下的 `random.py` 而不会加载 系统的 `random` 模块

1.4 原则 —— 每一个文件都应该是可以被导入的

- 一个 独立的 **`Python`** 文件 就是一个 模块
- 在导入文件时，文件中 所有没有任何缩进的代码 都会被从上到下执行一遍！

实际开发场景

- 在实际开发中，每一个模块都是独立开发的，大多都有专人负责
- 开发人员 通常会在 模块下方 增加一些测试代码
  - 仅在模块内使用，而被导入到其他文件中不需要执行

`__name__` 属性

- `__name__` 属性可以做到，测试模块的代码 只在测试情况下被运行，而在 被导入时不会被执行！
- `__name__` 是 `Python` 的一个内置属性，记录着一个 字符串
- 如果 是被其他文件导入的，`__name__` 就是 模块名
- 如果 是当前执行的程序 `__name__` 是 `__main__`

在很多 `Python` 文件中都会看到以下格式的代码：

```
# 导入模块
# 定义全局变量
# 定义类
# 定义函数

# 在代码的最下方
def main():
    # ...
    pass

# 根据 __name__ 判断是否执行下方代码
if __name__ == "__main__":
    main()
```

2. 包（Package）

概念

- 包 是一个 包含多个模块 的 特殊目录
- 目录下有一个 特殊的文件 `__init__.py`
- 包名的 命名方式 和变量名一致，小写字母 + `_`

好处

- 使用 `import 包名` 可以一次性导入 包 中 所有的模块

案例演练

1. 新建一个 `hm_message` 的 包
2. 在目录下，新建两个文件 `send_message` 和 `receive_message`
3. 在 `send_message` 文件中定义一个 `send` 函数
4. 在 `receive_message` 文件中定义一个 `receive` 函数
5. 在外部直接导入 `hm_message` 的包

`__init__.py`

- 要在外界使用 包 中的模块，需要在 `__init__.py` 中指定 对外界提供的模块列表

```
# 从 当前目录 导入 模块列表
from . import send_message
from . import receive_message
```

文件操作

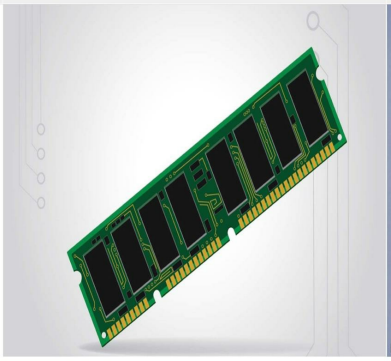
目标

1. 掌握文件的概念
2. 掌握文件的基本操作

1. 文件的概念

1.1 文件的概念和作用

- 计算机的 文件，就是存储在某种 长期储存设备 上的一段 数据
- 长期存储设备包括：硬盘、U 盘、移动硬盘、光盘...
- 作用：将数据长期保存下来，在需要的时候使用

CPU	内存	硬盘
		

1.2 文件的存储方式

- 在计算机中，文件是以 二进制 的方式保存在磁盘上的

文本文件和二进制文件

- 文本文件
  - 可以使用 文本编辑软件 查看
  - 本质上还是二进制文件
  - 例如：python 的源程序
- 二进制文件
  - 保存的内容 不是给人直接阅读的，而是 提供给其他软件使用的
  - 例如：图片文件、音频文件、视频文件等等
  - 二进制文件不能使用 文本编辑软件 查看

2. 文件的基本操作

2.1 操作文件的套路

在 计算机 中要操作文件的套路非常固定，一共包含三个步骤：

1. 打开文件
2. 读、写文件
   - 读 将文件内容读入内存
   - 写 将内存内容写入文件
3. 关闭文件

2.2 操作文件的函数/方法

在Python中要操作文件需要记住 1 个函数和 3 个方法

序号	函数/方法	说明
01	open	打开文件，并且返回文件操作对象
02	read	将文件内容读取到内存
03	write	将指定内容写入文件
04	close	关闭文件

- `open` 函数负责打开文件，并且返回文件对象
- `read` / `write` / `close` 三个方法都需要通过 文件对象 来调用

2.3 read 方法 —— 读取文件

- `open` 函数的第一个参数是要打开的文件名（文件名区分大小写）
  - 如果文件存在，返回文件操作对象
  - 如果文件不存在，会抛出异常
- `read` 方法可以一次性 读入 并 返回 文件的 所有内容
- `close` 方法负责 关闭文件
  - 如果 忘记关闭文件，会造成系统资源消耗，而且会影响到后续对文件的访问

方式一

```
# 1. 打开 - 文件名需要注意大小写
file = open("README")

# 2. 读取
text = file.read()
print(text)

# 3. 关闭
file.close()
```

该方式存在的问题：

- 如果在读取文件的过程中出现异常，则代码会终止执行，close()方法无法执行到，导致文件无法关闭
- 使用 `try ... finally` 实现代码比较繁琐，并且每次都要调用close()方法

方式二

使用 `with open()` 语句的好处就是在语句执行完后会自动关闭文件，即使出现异常。代码更加简洁，建议使用该方式

```
# 打开文件，并获取文件对象
with open("README") as f:
    # 读取
    text = f.read()
    print(text)
```

2.4 打开文件的方式

- `open` 函数默认以 只读方式 打开文件，并且返回文件对象

语法如下：

```
f = open("文件名", "访问方式")
```

访问方式	说明
r	以只读方式打开文件。这是默认模式。如果文件不存在，抛出异常
w	以只写方式打开文件。如果文件存在会被覆盖。如果文件不存在，创建新文件
a	以追加方式打开文件。如果该文件已存在，在文件末尾追加内容。如果文件不存在，创建新文件进行写入
rb	以二进制格式打开一个文件用于只读
wb	以二进制格式打开一个文件只用于写入

写入文件示例

```
with open("README", "w") as f:
    f.write("hello python! \n")
    f.write("今天天气真好")
```

2.5 按行读取文件内容

- `read` 方法默认会把文件的 所有内容 一次性读取到内存
- 如果文件太大，对内存的占用会非常严重

readline方法

- `readline` 方法可以一次读取一行内容
- 方法执行后，会把 文件指针 移动到下一行，准备再次读取

读取大文件的正确姿势

```
# 打开文件
with open("README") as file:
    while True:
        # 读取一行内容
        text = file.readline()

        # 判断是否读到内容
```

```
        if not text:
            break

        # 每读取一行的末尾已经有了一个 `\n`
        print(text, end="")
```

JSON操作

目标

1. 掌握JSON的语法格式
2. 熟练掌握对JSON数据的操作

1. JSON介绍

JSON的全称是”JavaScript Object Notation”，是JavaScript对象表示法，它是一种基于文本，独立于语言的轻量级数据交换格式。

1.1 JSON特点

- JSON是纯文本
- JSON具有良好的自我描述性，便于阅读和编写
- JSON具有清晰的层级结构
- 有效地提升网络传输效率

```
{
    "name": "tom",
    "age": 18,
    "isMan": true,
    "school": null,
    "address": {
        "country": "中国",
        "city": "江苏苏州",
        "street": "科技园路"
    },
    "numbers": [2, 6, 8, 9],
    "links": [
        {
            "name": "Baidu",
            "url": "http://www.baidu.com"
        },
        {
            "name": "TaoBao",
            "url": "http://www.taobao.com"
        }
    ]
}
```

JSON -> XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<person>
    <name>tom</name>
    <age>18</age>
    <isMan>true</isMan>
    <school />
    <address>
        <country>中国</country>
        <city>江苏苏州</city>
        <street>科技园路</street>
    </address>
    <numbers>2</numbers>
    <numbers>6</numbers>
    <numbers>8</numbers>
    <numbers>9</numbers>
    <links>
        <link>
            <name>Baidu</name>
            <url>http://www.baidu.com</url>
        </link>
        <link>
            <name>TaoBao</name>
            <url>http://www.taobao.com</url>
        </link>
    </links>
</person>
```

1.2 JSON语法规则

- 大括号保存对象
- 中括号保存数组
- 对象数组可以相互嵌套

- 数据采用键值对表示
- 多个数据由逗号分隔

JSON键

JSON的键必须是字符串类型，用英文双引号括起来

JSON值

JSON 值可以是：

- 数字（整数或浮点数）
- 字符串（在双引号中）
- 逻辑值（true 或 false）
- 数组（在中括号中）
- 对象（在大括号中）
- null

示例：

```
{
    "name": "tom",
    "age": 18,
    "isMan": true,
    "school": null,
    "address": {
        "country": "中国",
        "city": "江苏苏州",
        "street": "科技园路"
    },
    "numbers": [2, 6, 8, 9],
    "links": [
        {
            "name": "Baidu",
            "url": "http://www.baidu.com"
        },
        {
            "name": "TaoBao",
            "url": "http://www.taobao.com"
        }
    ]
}
```

---

2. JSON数据操作

本部分介绍如何对JSON文件进行读写操作，其中读取JSON文件需要重点掌握

2.1 导入依赖包

```
import json
```

2.2 JSON文件读写

读取**JSON**文件（重点）

```
with open('data.json', encoding='UTF-8') as f:
    data = json.load(f)  # 返回的数据类型为字典或列表
```

写入**JSON**文件

```
param = {'name': 'tom', 'age': 20}
with open('data2.json', 'w', encoding='UTF-8') as f:
    json.dump(param, f)
```