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

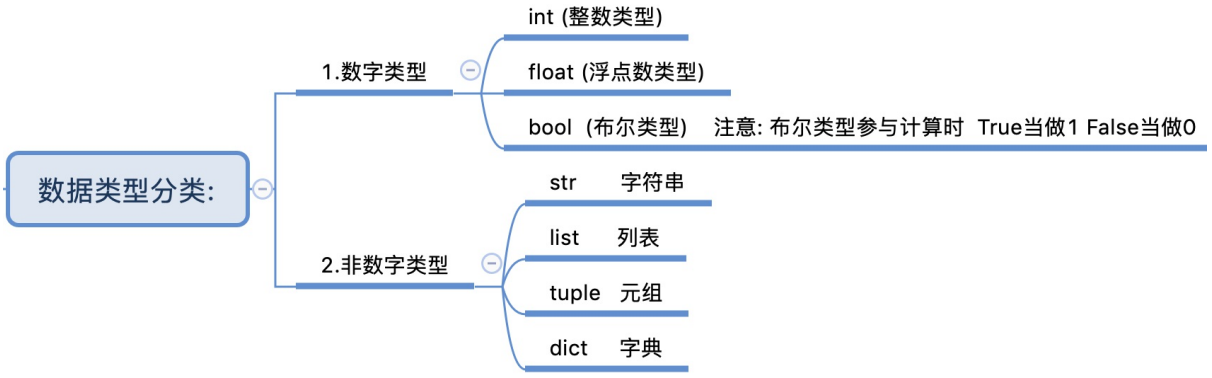
1. 掌握如何搭建Python开发环境；
2. 掌握Python基础语法, 具备基础的编程能力；
3. 建立编程思维以及面向对象程序设计思想；
4. 掌握如何通过UnitTest编写测试脚本，并生成HTML测试报告。

数据序列

目标

1. 掌握字符串常用操作
2. 掌握列表的作用和定义方式
3. 掌握列表常用的操作方法
4. 掌握元组常用的操作方法
5. 掌握字典的作用和定义方式
6. 掌握字典常用的操作方法

Python中数据类型分类



字符串

目标

1. 掌握字符串常用操作

1. 字符串基础

1.1 字符定义和使用

```
# 1. 自定义变量
# 2. 在变量中存储相应字符串
val1 = "黑马程序员软件测试"
val2 = '黑马程序员软件测试'

print(val1)
print(val2)
```

1.2 字符串定义细节

- 单引号字符串
- 双引号字符串
- 三引号字符串
- 特殊符号处理

```
# 1 单引号
val1 = '单引号软件测试'

# 2 双引号
val2 = "双引号软件测试"

# 3 三引号【也可以单双】
val3 = """三引号软件测试"""

# 4 转入字符串的特殊符号
val4 = "I\'m itcastYY"

# 5 去除特殊字符功能
val5 = r"I \' m itcastYY"
```

1.3 字符串下标访问

- 可以将字符串当做是装有很多内容的容器，通过编号可以获取到指定位置的字符
- 下标是人为定义的一种计数规则，默认下标从 0 开始
- 下标的使用语法为 `字符串名[下标值]`

```
# 下标获取对应位置上的字符
str1 = "夜晚来了我还依然睁着眼睛，是因为我看见了你留在月光下的痕迹"
print(str1[0])  # 夜
print(str1[1])  # 晚
```

2. 字符串切片

> 通过切片操作，可以获取字符串中指定部分的字符

2.1 切片语法

```
字符串[开始位置下标:结束位置下标:步长]
```

2.2 切片示例

```
name = "abcdefg"
print(name[2:5:1])  # cde
print(name[2:5])  # cde
print(name[:5])  # abcde
print(name[1:])  # bcdefg
print(name[:])  # abcdefg
print(name[::2])  # aceg
print(name[:-1])  # abcdef, 负1表示倒数第一个数据print(name[-4:-1])  # def
print(name[::-1])  # gfedcba
```

2.3 注意

- 结束下标位置对应的字符不会被截取到【俗称顾头不顾尾】
- 下标，正负都可以，不表示大小，只表示开始方向
- 步长用于设置截取间隔，默认步长为1

3. 字符串的常用操作-查找和替换

3.1 find

- 功能：被查找字符是否存在于当前字符串中，如果存在则返加开始下标，不存在则返回 -1
- 语法： `字符串.find(被查找字符，开始位置，结束位置 )`

示例：

```
mystr = "如果你给我的，和你给别人的是一样的，那我就不要了"
print(mystr.find('你'))  # 22
print(mystr.find('你', 3, 20))  # 8
print(mystr.find('你我'))  # -1
```

注意：

1. 开始位置和结束位置可以省略，表示从头找到尾
2. 如果被查找字符重复存在，则返回第一次出现位置的下标
3. 被查找字符如果不存在则返回 -1

3.2 replace

- 功能：使用新的子串，按规则替换旧的字符串内容
- 语法： `字符串.replace(原字符串，新子串，替换次数)`

示例：

```
str_val = "习近平主席我们称之为习大大，是受我们爱戴的伟人，金三胖同志被称之为世界最成功的80后"
new_str = str_val.replace("金三胖", "马赛克", 1)
print(str_val)
print(new_str)
```

注意：

1. 旧字符可能存在多次，此时可通过设置替换次数来决定具体替换多少个
2. 字符串属于不可变数据类型，所以修改并不会影响原来的空间

4. 字符串的常用操作-拆分和连接

4.1 split

- 功能：按照指定字符来分割字符串
- 语法： `字符串.split(分割符，num)`

示例：

```
mystr = "hello world and itcast and itheima and Python"

# 结果：['hello world ', ' itcast ', ' itheima ', ' Python']
print(mystr.split('and'))

# 结果：['hello world ', ' itcast ', ' itheima and Python']
print(mystr.split('and', 2))

# 结果：['hello', 'world', 'and', 'itcast', 'and', 'itheima', 'and', 'Python']
print(mystr.split(' '))

# 结果：['hello', 'world', 'and itcast and itheima and Python']
print(mystr.split(' ', 2))
```

注意：

1. num 为具体数值，当分割符存在多次时，可以设定使用几次
2. 默认不传入分割符则会以空格为边界

4.2 join

- 功能：一般用于将列表按指定子字符合并为字符串
- 语法： `字符串.join(一般为列表)`

示例：

```
list1 = ['张三', '李四', '小五', '黑']

print('_'.join(list1)) # 结果：张三_李四_小五_黑
```

列表

目标

1. 掌握列表的作用和定义方式
2. 掌握列表常用的操作方法

1. 列表基础

1.1 列表作用

- List（列表）是Python中使用最频繁的数据类型，在其他语言中通常叫做数组
- 专门用于存储一串信息（一组数据）

1.2 列表定义

- 方式一：用 `[]` 定义，数据之间使用英文逗号 `,` 分隔

```
name_list = []
name_list = ["zhangsan", "lisi", "wangwu"]
```

- 方式二：通过类实例化方式定义

```
data_list = list()
```

2. 列表操作-查询

2.1 下标

- 下标就是数据在列表中的位置编号，下标又可以被称为索引
- 列表的下标从0开始

```
name_list = ['张三', '李四']
print(name_list[0])  # 张三
print(name_list[1])  # 李四
```

2.2 count

- 功能： 统计被测试值出现的次数

- 语法： `列表.count(被测数据)`

示例：

```
list_val1 = ['飞蛾扑火时', '一定是', "极快乐幸福的", "一定是"]

print(list_val1.count("一定是"))  # 2
```

2.3 len

- 功能： 统计当前列表元素个数
- 语法： `len(列表)`

示例：

```
list_val1 = ['飞蛾扑火时', '一定是', "极快乐幸福的", "一定是"]

print(len(list_val1))  # 4
```

3. 列表操作-添加

3.1 append

- 功能：在列表的结尾添加数据
- 语法： `列表.append(被添加数据)`

示例：

```
val_list = ["Web自动化", "UI自动化", "接口自动化"]

val_list.append("APP自动化")

print(val_list)  # ['Web自动化', 'UI自动化', '接口自动化', 'APP自动化']
```

注意：

1. 使用 append() 将新的值添加在列表的末尾
2. 新增的值可以直接作用于原列表，固列表是可变数据类型
3. 如果使用append增加一个列表，则此列表会被当做一个值添加到末尾

3.2 extend

- 功能：在列表结尾添加另外一个列表，可以理解为是列表合并
- 语法： `列表.extend(列表2)`

示例：

```
num_list1 = [1, 2, 3]
num_list2 = [4, 5]
num_list1.extend(num_list2)
print(num_list1)  # [1, 2, 3, 4, 5]

name_list = ["张三", '李四']
name_list.extend("王五")
print(name_list)  # ['张三', '李四', '王', '五']
```

> 注：如果被添加值是不可变类型的可迭代对象，则会将其一一打散加入

3.3 insert

- 功能：在指定位置插入新数据
- 语法：`列表.insert(指定位置, 数据)`

示例：

```
val_list = ["Web自动化", "UI自动化", "接口自动化"]
val_list.insert(1, "APP自动化")

print(val_list)  # ['Web自动化', 'APP自动化', 'UI自动化', '接口自动化']
```

4. 列表操作-删除

4.1 pop

- 功能：删除指定下标的数据，并且返回被删除数据
- 语法：`列表.pop(下标)`

示例：

```
val_list = ["Web自动化", "UI自动化", "接口自动化"]

val = val_list.pop(0)

print(val, val_list)  # web自动化, ['UI自动化', '接口自动化']
```

注意：

1. `pop()` 方法不传递参数时，默认删除列表中最后一个数据
2. pop() 方法有返回值，会返回当前被删除的值

4.2 remove

- 功能：在列表中删除某个具体数据，删除第一个被匹配到的
- 语法：`列表.remove(数据)`

示例：

```
val_list = ["Web自动化", "UI自动化", "接口自动化", "Web自动化"]

val = val_list.remove("Web自动化")

print(val, val_list)  # None ['UI自动化', '接口自动化', 'Web自动化']
```

注意：

1. remove 方法使用不会返回被删除的值
2. 如果被删除数据出现多次，只会删除第一次出现的

4.3 clear

- 功能：清空列表
- 语法：`列表.clear()`

示例：

```
val_list = ["Web自动化", "UI自动化", "接口自动化", "Web自动化"]
val_list.clear()

print(val_list)  # []
```

5. 列表操作-修改

5.1 下标修改

- 功能：通过指定下标修改对应数据
- 语法：`列表[下标] = 修改后的值`

示例：

```
val_list = ["Web自动化", "UI自动化", "接口自动化", "Web自动化"]

val_list[1] = "黑马程序员"

print(val_list)  # ['Web自动化', '黑马程序员', '接口自动化', 'Web自动化']
```

5.2 reverse

- 功能：倒置列表，列表反转
- 语法：`列表.reverse()`

示例：

```
num_list = [1, 2, 3, 4]
num_list.reverse()
print(num_list)  # [4, 3, 2, 1]
```

5.3 排序

- 功能：将列表按指定规则进行数据排序
- 语法：`列表.sort(key=None, reverse=False)`

示例：

```
val_list = [8, 100, 30, 10, 40, 2]

val_list.sort(reverse=True)

print(val_list)  # [100, 40, 30, 10, 8, 2]
```

注意：reverse 表示排序规则，默认是False表示升序，设置为True表示降序

6. 列表其它操作

6.1 复制

- 语法：`列表.copy()`

示例：

```
num_list1 = [1, 2, 3]
num_list2 = num_list1.copy()

num_list1.append(4)
print(num_list1)  # [1, 2, 3, 4]
print(num_list2)  # [1, 2, 3]
```

6.2 for循环遍历

使用for语法实现列表的遍历

```
val_list = [8, 100, 30, 10, 40, 2]

for item in val_list:
    print(item)
```

6.3 列表嵌套

- 列表可以多层嵌套
- 不论多少层都可以使用下标进行访问

```
person_info = [["张三", "18", "功能测试"], ["李四", "20", "自动化测试"]]
print(person_info[0][1])  # 18
```

元组

目标

1. 掌握元组常用的操作方法

1. 元组基础

1.1 元组的作用

1. 元组和列表一样，都可用于存储多个数据
2. 有些数据在存储之后就不能发生改变
3. 通过元组可以存放多个数据，且这些数据不能被修改

1.2 元组的定义

- 方式一：用 `()` 定义，数据之间使用英文逗号 `,` 分隔

```
info_tuple = ()
info_tuple = ("zhangsan", 18, 1.75)
```

- 方式二：通过类实例化方式定义

```
info_tuple = tuple()
```

注意：元组中只包含一个元素时，需要在元素后面添加逗号

```
# 只有一个值的元组
data = (1,)
```

2. 元组常见操作

2.1 下标查找元素

- 功能：返回指定下标对应的元素
- 语法：`元组[下标]`

示例：

```
tuple1 = (1, 2, 3)
```

```
print(tuple1[1])  # 2
```

2.2 count

- 功能：统计某个数据出现的次数
- 语法：`列表.count(被测数据)`

示例：

```
tuple1 = (1, 2, 3)
print(tuple1.count(3))  # 1
```

2.3 len

- 功能：返回目标元组的长度
- 语法：`len(元组)`

示例：

```
tuple1 = (1, 2, 3)

print(len(tuple1))  # 3
```

字典

目标

1. 掌握字典的作用和定义方式
2. 掌握字典常用的操作方法

1. 字典基础

1.1 字典的作用

> 思考：会员管理系统需要处理不同会员身份信息，如何区分不同会员？同一个会员如何明确年纪、身高、体重等数据？

- 实际业务很复杂， 需要用到不同类型的数据
- 不同类型数据在保存时应当加以区分
- 字典不仅可以保存多个数据，同时还能给不同数据 ”起名字“

1.2 字典的定义

1. 基本结构： `字典名 = {}`
2. 大括号内结构为： `键名：键值` 【俗称键值对】
3. 多个键值对之间使用逗号隔开

```
user_info = {
    "name": "xiaoming",
    "age": 28,
    "gender": "man"
}
```

2. 字典常见操作

2.1 增

- 功能：在字典中增加对应的键值对
- 语法： `字典名['键名'] = 新增键值`

示例：

```
info = {
```

```
    "name": "syy",
    "age": 18,
    "gender": "man"
}
info["salary"] = 100000

print(info)  # {'name': 'syy', 'age': 18, 'gender': 'man', 'salary': 100000}
```

2.2 删

- 功能：删除指定字典或字典中的值
- 语法：`del 字典[key]`

示例：

```
info = {
    "name": "syy",
    "age": 18,
    "gender": "man"
}

# 删除值
del info["age"]
print(info)
```

2.3 清空

- 功能：清空整个字典
- 语法：`字典.clear()`

示例：

```
info = {
    "name": "syy",
    "age": 18,
    "gender": "man"
}
info.clear()

print(info)  # {}
```

2.4 修改

- 功能：修改字典中对应键的值
- 语法：`字典["键名"] = val`

示例：

```
info = {
    "name": "syy",
    "age": 18,
    "gender": "man"
}
info["age"] = 28

print(info)
```

注：

1. 如果被修改的键名存在，则直接用新值更新旧值
2. 如果被修改的键名不存在，则相当于新增

2.5 键名查找

- 功能：通过键名来查找对应的值
- 语法：`字典名["键名"]`

示例：

```
slogan = {
    "slogan1": "尘归于尘,土归于土,我，归于我们",
    "slogan2": "有时候我们要对自己残忍一点，不能纵容自己的伤心失望;有时候我们要对自己深爱的人残忍一点，将对他们的爱的记忆搁置",
    "slogan3": "红尘十丈，茫茫的人海，竟还是自己的来处"
}

print(slogan["slogan3"])
```

> 注意：此种方法查找时，如果键名不存在则会语法报错

2.6 get查找

- 功能： 返回对应键名的值
- 语法：`字典名.get(key, 默认值)`

示例：

```
slogan = {
    "slogan1": "尘归于尘,土归于土,我，归于我们",
    "slogan2": "有时候我们要对自己残忍一点，不能纵容自己的伤心失望;有时候我们要对自己深爱的人残忍一点，将对他们的爱的记忆搁置",
    "slogan3": "红尘十丈，茫茫的人海，竟还是自己的来处"
```

```
}

print(slogan.get("abc"))
```

> 注意：如果键名不存在，不会抛出异常，而是返回None

3. 字典遍历操作

3.1 遍历字典的Key

- 功能：循环拿到字典中的每个键名
- 语法：使用循环结构完成

示例：

```
slogan = {
    "slogan1": "尘归于尘,土归于土,我，归于我们",
    "slogan2": "有时候我们要对自己残忍一点，不能纵容自己的伤心失望",
    "slogan4": "有时候我们要对自己深爱的人残忍一点，将对他们的爱的记忆搁置",
    "slogan3": "红尘十丈，茫茫的人海，竟还是自己的来处"
}

for key in slogan.keys():
    print(key)
# slogan1 slogan2 slogan3 slogan4
```

3.2 遍历字典的Value

- 功能：循环拿到每个键对应的值
- 语法：使用循环结构

示例：

```
slogan = {
    "slogan1": "尘归于尘,土归于土,我，归于我们",
    "slogan2": "有时候我们要对自己残忍一点，不能纵容自己的伤心失望",
    "slogan4": "有时候我们要对自己深爱的人残忍一点，将对他们的爱的记忆搁置",
    "slogan3": "红尘十丈，茫茫的人海，竟还是自己的来处"
}

for key in slogan.values():
    print(key)
```

3.3 遍历字典的Key和Value

调用字典的 `items()` 方法获取字典的键和值，并自动赋值给不同的变量

示例：

```
user = {
    "name": "张三",
    "age": 18,
    "gender": "男",
}
for k, v in user.items():
    print("Key=%s Value=%s" % (k, v))

# Key=name Value=张三
# Key=age Value=18
# Key=gender Value=男
```