

# 暨南大学本科实验报告专用纸

课程名称 python程序设计 成绩评定           

实验项目名称 python作业第九章

姓名 崔嘉容 学号 2020100069 学院 网络空间安全 专业 网络空间安全

实验时间 2023 年 11 月 13 日 ~ 11 月 20 日 实验地点: 516

## 一、实验目的

1. 理解面向对象的基本概念。
2. 掌握类的定义方法，根据具体需求设计类。
3. 了解如何定义类的私有数据成员和成员方法。
4. 掌握如何使用自定义类实例化对象。

## 二、实验环境和设备

实验环境：操作系统-Windows10，python版本-3.11.3，开发环境-pycharm

实验设备：华为MateBook14-2020，处理器-i7-10510U，内存-16GB

## 三、实验内容和结果

**题目一：**定义一个学生类，包括学号、姓名、出生日期三个属性（数据成员）；包括一个用于给定数据成员初始值的构造函数；包含一个计算学生年龄的方法。编写该类并对其进行测试。

**实验代码：**

```
from datetime import datetime
class Student:
    def __init__(self, stu_id, name, birth_day):
        # 构造函数，用于初始化学生对象的属性
        self.stu_id=stu_id
        self.name=name
        self.birth_day=birth_day

    def calculate_age(self):
        # 计算学生年龄的方法
```

# 暨南大学本科实验报告专用纸(附页)

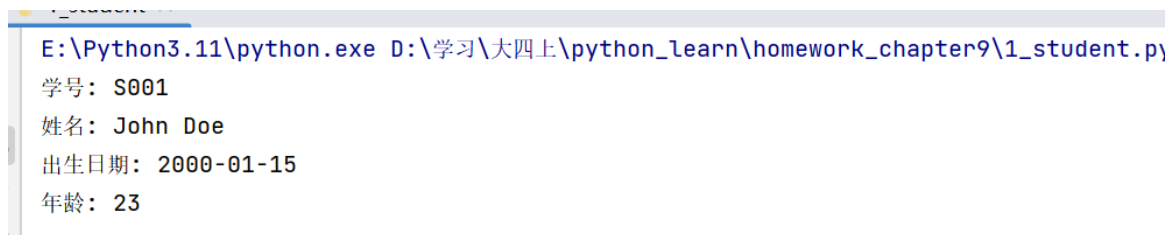
```
today = datetime.today() # 获取当前日期时间
birth_day = datetime.strptime(self.birth_day, "%Y-%m-%d") # 将
出生日期字符串转换为 datetime 对象
age = today.year - birth_day.year - ((today.month, today.day) <
(birth_day.month, birth_day.day))
return age

if __name__ == "__main__":
    # 创建一个学生对象
    student1 = Student(stu_id="S001", name="John Doe", birth_day="2000-
01-15")

    # 输出学生信息
    print("学号:", student1.stu_id)
    print("姓名:", student1.name)
    print("出生日期:", student1.birth_day)

    # 计算并输出学生年龄
    age = student1.calculate_age()
    print("年龄:", age)
```

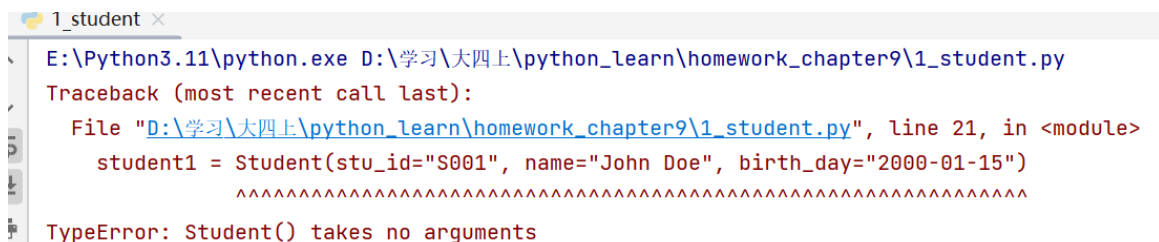
## 实验结果:



```
E:\Python3.11\python.exe D:\学习\大四上\python_learn\homework_chapter9\1_student.py
学号: S001
姓名: John Doe
出生日期: 2000-01-15
年龄: 23
```

## 实验分析和总结:

一开始出现这个报错:



```
E:\Python3.11\python.exe D:\学习\大四上\python_learn\homework_chapter9\1_student.py
Traceback (most recent call last):
  File "D:\学习\大四上\python_learn\homework_chapter9\1_student.py", line 21, in <module>
    student1 = Student(stu_id="S001", name="John Doe", birth_day="2000-01-15")
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
TypeError: Student() takes no arguments
```

查询后明白该错误信息, 提示表明在创建Student对象时出现了类型错误 (TypeError)。具体地说, 它表明Student类的构造函数不接受提供的参数。检查代码后发现错误的将构造函数的名称应该是\_\_init\_\_而不是\_\_int\_\_, 修改过后程序不再报错。

# 暨南大学本科实验报告专用纸(附页)

**题目二:** 定义一个shape类, 利用它作为基类派生出Rectangle、Circle等据图形状, 已知具体形状类均具有两个方法GetArea和GetColor, 分别用来得到形状的面积和颜色。

## 实验代码:

```
import math

class Shape:
    def __init__(self, color):
        self.color = color

    def GetArea(self):
        return self.area

    def GetColor(self):
        return self.color

class Rectangle(Shape):
    def __init__(self, color, width, height):
        super().__init__(color)
        self.width = width
        self.height = height

    def get_area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, color, radius):
        super().__init__(color)
        self.radius = radius

    def get_area(self):
        return math.pi * self.radius ** 2

# 测试
if __name__ == "__main__":
    # 创建一个矩形对象
    rectangle = Rectangle(color="Blue", width=4, height=5)

    # 输出矩形的面积和颜色
    print("矩形面积:", rectangle.get_area())
    print("矩形颜色:", rectangle.GetColor())

    # 创建一个圆形对象
    Python程序设计课程实验报告
```

# 暨南大学本科实验报告专用纸(附页)

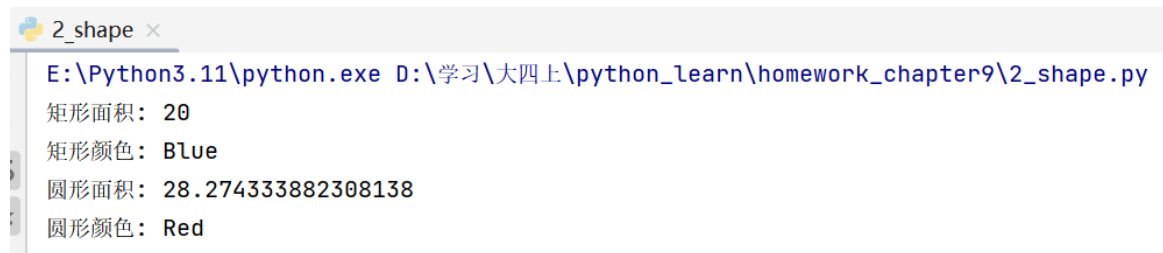
```
circle = Circle(color="Red", radius=3)
```

```
# 输出圆形的面积和颜色
```

```
print("圆形面积:", circle.get_area())
```

```
print("圆形颜色:", circle.GetColor())
```

## 实验结果:



```
2_shape x
E:\Python3.11\python.exe D:\学习\大四上\python_learn\homework_chapter9\2_shape.py
矩形面积: 20
矩形颜色: Blue
圆形面积: 28.274333882308138
圆形颜色: Red
```

## 实验分析和总结:

```
def GetArea(self):
```

```
    return self.area
```

这个方法是一个具体的实现，它返回对象的属性 `self.area` 的值。这意味着在调用这个方法时，它会直接返回对象中存储的 `area` 属性的值。这种实现方式适用于那些在对象创建时就已经计算好并存储在属性中的情况。

```
def get_area(self):
```

```
    pass # 子类需要实现具体的计算面积的方法
```

第二个方法是一个抽象方法的框架，其中包含了 `pass` 语句。`pass` 是一个占位符，表示这个方法暂时不做任何事情，它需要在派生类中进行具体的实现。在这种情况下，`pass` 用于提醒程序员在派生类中实现具体的计算面积的方法，因为在基类中不可能知道具体的实现细节。

在本题目中，由于 `GetArea` 是一个用于计算面积的方法，最好选择第二种方法，即在基类 `Shape` 中使用抽象方法框架。这样可以鼓励在派生类中实现具体的面积计算方法，以适应不同形状的要求。

# 暨南大学本科实验报告专用纸(附页)

## 题目三:

定义一个 stack 类, 要求实现栈的以下操作: ↵

操作↵	含义↵
push (element) ↵	将数据存入栈↵
pop↵	将数据取出栈↵
empty↵	判断栈是否为空↵
full↵	判断栈是否满↵
find(element)↵	查找元素返回位置↵
peek↵	返回栈顶元素↵

## 实验代码:

```
class Stack:
    def __init__(self, maxsize):
        self.maxsize = maxsize
        self.items = []

    # 判满
    def Isfull(self):
        return len(self.items) == self.maxsize

    # 判空
    def Isempy(self):
        return len(self.items) == 0

    # 入栈
    def Push(self, item):
        if not self.Isfull():
            self.items.append(item)
        else:
            print("栈满, 入栈失败!")

    # 出栈
    def Pop(self):
        if not self.Isempy():
            pop_item = self.items.pop()
            return pop_item
```

# 暨南大学本科实验报告专用纸(附页)

```
else:
    print("栈已空, 无法出栈")

# 查找位置
def Find(self, item):
    if item in self.items:
        index = len(self.items) - 1 - self.items[::-1].index(item)
        print(f"{item} 的位置在栈中的第 {index + 1} 个位置 (从栈底开始计数)")
    else:
        print(f"{item} 不在栈中")

# 返回栈顶
def Peek(self):
    if not self.IsEmpty():
        return self.items[-1]
    else:
        print("栈已空, 无栈顶元素")

# 显示栈内元素
def Display(self):
    print("栈中元素:", self.items)

# 测试
if __name__ == "__main__":
    stack = Stack(maxsize=5)

    stack.Push(1)
    stack.Push(2)
    stack.Push(3)
    stack.Display()

    stack.Pop()
    stack.Display()

    stack.Push('a')
    stack.Push('b')
    stack.Push('c')
    stack.Display()

    stack.Push("123") # 测试栈已满的情况
    print("栈顶元素:", stack.Peek())

    stack.Find('a')
    stack.Find(6) # 测试查找不在栈中的元素
```

# 暨南大学本科实验报告专用纸(附页)

## 实验结果:

```
3_stack x
E:\Python3.11\python.exe D:\学习\大四上\python_learn\homework_chapter9\3_stack.py
栈中元素: [1, 2, 3]
栈中元素: [1, 2]
栈中元素: [1, 2, 'a', 'b', 'c']
栈满, 入栈失败!
栈顶元素: c
a 的位置在栈中的第 3 个位置 (从栈底开始计数)
6 不在栈中
```

## 实验分析和总结:

在 `pop` 方法中使用了 `remove` 方法而不是 `pop`, 导致不正确地删除了列表中的元素。`pop` 方法是列表对象的一个方法, 用于移除列表中指定位置的元素, 并返回该元素的值。如果不指定索引, 默认移除并返回列表中的最后一个元素。在栈的实现中, `pop` 方法通常被用于移除栈顶元素, 因为它移除的是列表中的最后一个元素。

`remove` 方法是列表对象的另一个方法, 用于移除列表中第一个匹配给定值的元素, `remove` 方法会删除第一个匹配的元素, 而不是指定位置的元素。在栈的实现中, 如果误用 `remove` 来删除元素, 会导致不正确地删除列表中的一个匹配元素, 而不是栈顶元素。

`self.items.remove()` 会删除列表中的第一个元素, 并返回 `None`, 而不是栈顶元素。这会导致在调用 `pop` 方法后, `popped_item` 变量的值是 `None`, 而不是栈顶元素的值。这样的错误使用可能会导致程序逻辑混乱, 不符合栈的出栈操作的语义。正确的做法是使用 `pop` 方法。

通过这次实验, 深入理解了栈的基本原理和实现, 学会了如何用类的形式抽象出栈的概念, 并通过类的方法实现栈的各项操作。这种面向对象的设计有助于提高代码的可维护性和可扩展性, 使得栈的实现可以轻松地应用于不同的场景。在实际应用中, 可以根据具体需求对 `Stack` 类进行扩展和修改。总体而言, 这次实验为提供了一个良好的实践机会, 更好地掌握了面向对象编程和数据结构的知识。

# 暨南大学本科实验报告专用纸(附页)

**题目四:** 定义一个音乐播放器类，该类必须包含的类成员（数据成员）有歌名和歌手（其他的属性可以自行添加），并实现播放、暂停、切歌等功能。（注可以使用文字模拟，也可以使用相关函数真实的播放歌曲）。

## 实验代码:

```
import tkinter as tk
from tkinter import ttk
from pygame import mixer # 导入 pygame 库

class MusicPlayer:
    def __init__(self, root):
        self.root = root
        self.root.title("音乐播放器")

        # 数据成员
        self.current_song = tk.StringVar()
        self.current_artist = tk.StringVar()
        self.playing = False

        # 创建界面元素
        self.song_label = tk.Label(root, textvariable=self.current_song,
font=("Helvetica", 16))
        self.artist_label = tk.Label(root,
textvariable=self.current_artist, font=("Helvetica", 12))
        self.progress_bar = ttk.Progressbar(root, orient="horizontal",
length=300, mode="determinate")
        self.play_button = tk.Button(root, text="播放",
command=self.toggle_play)
        self.next_button = tk.Button(root, text="切歌",
command=self.next_song)

        # 布局
        self.song_label.pack(pady=10)
        self.artist_label.pack(pady=5)
        self.progress_bar.pack(pady=10)
        self.play_button.pack(side="left", padx=10)
        self.next_button.pack(side="right", padx=10)

        # 模拟歌曲数据
        self.songs = [("song1.mp3", "歌手 1"), ("song2.mp3", "歌手 2"),
("song3.mp3", "歌手 3")]
```



# 暨南大学本科实验报告专用纸(附页)

```
self.current_song_index = 0

# 初始化界面
self.update_song_info()

def toggle_play(self):
    self.playing = not self.playing
    if self.playing:
        self.play_button["text"] = "暂停"
        self.play_music()
    else:
        self.play_button["text"] = "播放"
        self.pause_music()

def play_music(self):
    song_path, _ = self.songs[self.current_song_index]
    mixer.init() # 初始化 mixer
    mixer.music.load(song_path)
    mixer.music.play()

def pause_music(self):
    mixer.music.pause()

def next_song(self):
    mixer.music.stop()
    self.current_song_index = (self.current_song_index + 1) %
len(self.songs)
    self.update_song_info()
    self.play_music()

def update_song_info(self):
    current_song, current_artist =
self.songs[self.current_song_index]
    self.current_song.set(f"歌名: {current_song}")
    self.current_artist.set(f"歌手: {current_artist}")

# 创建主窗口
root = tk.Tk()
app = MusicPlayer(root)
root.mainloop()
```

## 实验结果:

# 暨南大学本科实验报告专用纸(附页)



## 实验分析和总结:

### 1. 代码结构和功能

该代码实现了一个简单的音乐播放器界面，使用了 `tkinter` 进行 GUI 开发，并利用 `pygame` 库处理音乐播放的功能。主要功能包括：播放/暂停音乐、切换歌曲、显示当前歌曲信息（歌名、歌手）、使用进度条显示歌曲播放进度。

### 2. 使用的库和模块

`tkinter`：用于创建图形用户界面。`ttk.Progressbar`：提供了进度条控件，用于显示歌曲播放进度。`pygame.mixer`：提供了音乐播放相关的功能。

### 3. 代码分析

初始化界面元素：在 `__init__` 方法中，通过创建 `Label`、`Progressbar`、`Button` 等界面元素，实现了音乐播放器的基本布局。模拟歌曲数据：使用 `self.songs` 列表存储模拟的歌曲数据，每个歌曲是一个元组，包含歌曲文件路径和歌手信息。播放/暂停音乐：`toggle_play` 方法通过切换 `playing` 属性实现播放和暂停功能，调用 `play_music` 和 `pause_music` 方法处理具体的音乐播放逻辑。

切换歌曲：`next_song` 方法停止当前歌曲播放，更新当前歌曲索引，然后调用 `update_song_info` 和 `play_music` 方法切换到下一首歌曲。更新歌曲信息：`update_song_info` 方法根据当前歌曲索引更新界面上的歌曲信息。

### 4. 实验总结

代码中的异常处理：在实际应用中，可以考虑添加更多的异常处理机制，例如处理文件选择对话框取消选择的情况，处理歌曲播放过程中的异常等。

用户体验：该音乐播放器还可以进一步改进，例如添加音量控制、随机播放、循环播放等功能，以提升用户体验。

使用第三方库：`pygame` 被用于音乐播放功能，这是一个专业的多媒体库，适合处理音频、视频等。在实际项目中，可以考虑使用更专业的音乐播放库，以提供更多的功能和更好的性能。