Python 入门讲义

目录

1. 简介

Python 是一种高级、通用、解释型编程语言。它的设计哲学强调代码的可读性，使用缩进作为代码块的结构。Python 支持多种编程范式，包括面向对象、命令式、函数式和过程式编程。由于其简洁的语法和庞大的标准库，Python 被广泛应用于 Web 开发、数据科学、人工智能、脚本编写和自动化等领域。

Python 的优点：

- **易于学习:** Python 语法简洁清晰，非常适合初学者入门。
- **可读性强:** Python 代码具有很高的可读性，易于理解和维护。
- **庞大的标准库:** Python 拥有丰富的标准库，提供了各种各样的模块和函数，可以完成许多常见的任务。
- **强大的第三方库:** Python 社区非常活跃，提供了大量的第三方库，可以扩展 Python 的功能，例如 NumPy (数值计算), Pandas (数据分析), Matplotlib (绘图), Django/Flask (Web 开发), TensorFlow/PyTorch (机器学习) 等。
- **跨平台性:** Python 可以在多种操作系统上运行，包括 Windows, macOS, Linux 等。
- **解释型语言:** Python 代码不需要编译，可以直接运行，方便快捷。

2. 基础语法

2.1 变量和数据类型

变量: 变量是用于存储数据的命名存储位置。在 Python 中，您不需要显式声明变量的类型，Python 会根据您赋给变量的值自动推断类型。

```
# 变量赋值
x = 10          # 整数
y = 3.14        # 浮点数
name = "Alice"  # 字符串
is_student = True # 布尔值

# 打印变量的值
print(x)
print(y)
print(name)
print(is_student)
```

数据类型: Python 中常用的数据类型包括：

- **整数 (int):** 例如: `10`, `-5`, `0`
- **浮点数 (float):** 例如: `3.14`, `-2.5`, `0.0`
- **字符串 (str):** 例如: `"Hello"`, `'Python'`, `"123"`
- **布尔值 (bool):** `True` 或 `False`
- **列表 (list):** 有序可变序列，例如: `[1, 2, 3]`, `['a', 'b', 'c']`
- **元组 (tuple):** 有序不可变序列，例如: `(1, 2, 3)`, `('x', 'y', 'z')`
- **字典 (dict):** 键值对的集合，例如: `{'name': 'Bob', 'age': 30}`
- **集合 (set):** 无序不重复元素的集合，例如: `{1, 2, 3}`, `{'apple', 'banana', 'orange'}`

可以使用 `type()` 函数来查看变量的数据类型：

```
print(type(x))          # <class 'int'>
print(type(y))          # <class 'float'>
print(type(name))       # <class 'str'>
print(type(is_student)) # <class 'bool'>
```

2.2 运算符

Python 支持各种运算符，包括：

- **算术运算符:** + (加), - (减), * (乘), / (除), // (整除), % (取余), ** (幂)
- **比较运算符:** == (等于), != (不等于), > (大于), < (小于), >= (大于等于), <= (小于等于)
- **赋值运算符:** = (赋值), +=, -=, *=, /=, //=, %=, **=
- **逻辑运算符:** and (与), or (或), not (非)
- **成员运算符:** in (在…中), not in (不在…中)
- **身份运算符:** is (是), is not (不是)

```
a = 10
b = 5

# 算术运算符
print(a + b)   # 15
print(a - b)   # 5
print(a * b)   # 50
print(a / b)   # 2.0
print(a // b)  # 2
print(a % b)   # 0
print(a ** b)  # 100000

# 比较运算符
print(a == b)  # False
print(a != b)  # True
print(a > b)   # True
```

```
print(a < b)   # False

# 逻辑运算符
c = True
d = False
print(c and d) # False
print(c or d)  # True
print(not c)   # False
```

2.3 注释

注释用于在代码中添加说明，提高代码的可读性。Python 中使用 # 符号进行单行注释，使用 ''' 或 """ 进行多行注释。

```
# 这是一个单行注释

'''
这是一个
多行注释
'''

"""
这也是一个
多行注释
"""

print("Hello, World!") # 这行代码打印 "Hello, World!"
```

2.4 基本输入/输出

- **输出:** 使用 `print()` 函数将信息输出到控制台。

```
print("Hello, Python!")
print("The value of x is:", x)
```

- **输入:** 使用 `input()` 函数从控制台接收用户输入。`input()` 函数返回的是字符串类型。

```
name = input(" 请输入你的名字: ")
print(" 你好,", name)

age_str = input(" 请输入你的年龄: ")
age = int(age_str) # 将字符串转换为整数
print(" 你的年龄是:", age)
```

3. 数据结构

3.1 列表 (List)

列表是 Python 中最常用的数据结构之一。它是一个有序可变的元素集合，可以包含不同类型的元素。

```
# 创建列表
my_list = [1, 2, 3, 'apple', 'banana']
print(my_list)

# 访问列表元素 (索引从 0 开始)
print(my_list[0])   # 1
print(my_list[3])   # 'apple'
print(my_list[-1])  # 'banana' (最后一个元素)

# 修改列表元素
my_list[1] = 10
print(my_list)      # [1, 10, 3, 'apple', 'banana']
```

```
# 列表切片（获取子列表）
print(my_list[1:4])  # [10, 3, 'apple']（从索引 1 到 3，不包含索引 4）
print(my_list[:3])   # [1, 10, 3]（从开始到索引 2）
print(my_list[2:])   # [3, 'apple', 'banana']（从索引 2 到结束）

# 列表常用方法
my_list.append('orange') # 在列表末尾添加元素
print(my_list)      # [1, 10, 3, 'apple', 'banana', 'orange']

my_list.insert(2, 'grape') # 在指定索引位置插入元素
print(my_list)      # [1, 10, 'grape', 3, 'apple', 'banana', 'orange']

my_list.remove('apple') # 删除指定元素（只删除第一个匹配项）
print(my_list)      # [1, 10, 'grape', 3, 'banana', 'orange']

popped_item = my_list.pop(1) # 删除指定索引位置的元素并返回该元素（默认删除最后一个元素）
print(popped_item)  # 10
print(my_list)      # [1, 'grape', 3, 'banana', 'orange']

print(len(my_list)) # 获取列表长度（元素个数）
```

3.2 元组 (Tuple)

元组与列表类似，也是有序的元素集合，**但是元组是不可变的**，一旦创建就不能修改。

```
# 创建元组
my_tuple = (1, 2, 3, 'a', 'b')
print(my_tuple)

# 访问元组元素（索引从 0 开始）
print(my_tuple[0])   # 1
```

```
print(my_tuple[3])   # 'a'

# 元组切片
print(my_tuple[1:4])  # (2, 3, 'a')

# 元组是不可变的，不能修改元素
# my_tuple[0] = 10  # 会报错 TypeError: 'tuple' object does not support item assignment

# 元组常用方法 (元组的方法较少，因为不可变)
print(my_tuple.count(2)) # 统计元素 2 出现的次数
print(my_tuple.index('a')) # 查找元素 'a' 的索引 (第一个匹配项)
```

3.3 字典 (Dictionary)

字典是键值对 (key-value pairs) 的集合。键 (key) 必须是唯一的且不可变的 (例如字符串、数字、元组)，值 (value) 可以是任意类型。字典是无序的。

```
# 创建字典
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}
print(my_dict)

# 访问字典的值 (通过键)
print(my_dict['name'])  # 'Alice'
print(my_dict['age'])   # 25

# 修改字典的值
my_dict['age'] = 26
print(my_dict)      # {'name': 'Alice', 'age': 26, 'city': 'New York'}

# 添加新的键值对
my_dict['gender'] = 'female'
```

```
print(my_dict)      # {'name': 'Alice', 'age': 26, 'city': 'New York', 'gender': 'female'}

# 删除键值对
del my_dict['city']
print(my_dict)      # {'name': 'Alice', 'age': 26, 'gender': 'female'}

# 字典常用方法
print(my_dict.keys())   # 获取所有键 (返回一个视图对象)
print(my_dict.values()) # 获取所有值 (返回一个视图对象)
print(my_dict.items())  # 获取所有键值对 (返回一个视图对象, 每个元素是 (key, value) 元组)

print('name' in my_dict) # 检查键 'name' 是否存在于字典中

print(my_dict.get('name'))     # 获取键 'name' 对应的值, 如果键不存在返回 None
print(my_dict.get('country', 'Unknown')) # 获取键 'country' 对应的值, 如果键不存在返回默认值
```

3.4 集合 (Set)

集合是无序且不重复元素的集合。集合主要用于成员关系测试和删除重复元素。

```
# 创建集合
my_set = {1, 2, 3, 3, 4, 5} # 重复元素会自动去重
print(my_set)      # {1, 2, 3, 4, 5}

my_set2 = set([4, 5, 6, 7]) # 使用 list 创建集合
print(my_set2)     # {4, 5, 6, 7}

# 集合常用操作
my_set.add(6)      # 添加元素
print(my_set)      # {1, 2, 3, 4, 5, 6}

my_set.remove(1)   # 删除元素 (如果元素不存在会报错 KeyError)
```

```
print(my_set)      # {2, 3, 4, 5, 6}

my_set.discard(7)  # 删除元素（如果元素不存在不会报错）
print(my_set)      # {2, 3, 4, 5, 6}

print(3 in my_set)  # 检查元素 3 是否在集合中

# 集合运算
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

print(set1 | set2)  # 并集 {1, 2, 3, 4, 5, 6}
print(set1 & set2)  # 交集 {3, 4}
print(set1 - set2)  # 差集 {1, 2}（set1 中有但 set2 中没有的元素）
print(set2 - set1)  # 差集 {5, 6}（set2 中有但 set1 中没有的元素）
print(set1 ^ set2)  # 对称差集 {1, 2, 5, 6}（只在一个集合中出现的元素）
```

4. 控制语句

4.1 条件语句 (if, elif, else)

条件语句用于根据条件执行不同的代码块。

```
age = 20

if age >= 18:
    print(" 成年人")
elif age >= 13:
    print(" 青少年")
else:
    print(" 儿童")
```

- `if` 语句开始一个条件判断。
- `elif` (else if) 语句用于检查额外的条件，可以有多个 `elif`。
- `else` 语句在所有 `if` 和 `elif` 条件都不满足时执行。

4.2 循环语句

- **for 循环:** 用于遍历序列 (列表、元组、字符串) 或其他可迭代对象。

```
# 遍历列表
fruits = ['apple', 'banana', 'orange']
for fruit in fruits:
    print(fruit)

# 遍历字符串
for char in "Python":
    print(char)

# 使用 range() 生成数字序列
for i in range(5): # range(5) 生成 0, 1, 2, 3, 4
    print(i)

for i in range(1, 10, 2): # range(start, stop, step)
    print(i) # 1, 3, 5, 7, 9
```

- **while 循环:** 只要条件为真，就一直执行循环体。

```
count = 0
while count < 5:
    print(count)
    count += 1

# break 语句用于跳出循环
```

```
while True:
    name = input(" 请输入你的名字 (输入 'quit' 退出): ")
    if name == 'quit':
        break
    print(" 你好,", name)

# continue 语句用于跳过当前循环迭代，继续下一次迭代
for i in range(10):
    if i % 2 == 0:
        continue # 如果 i 是偶数，跳过本次迭代
    print(i) # 只打印奇数
```

5. 函数

函数是组织好的、可重复使用的代码块，用于执行特定任务。函数可以提高代码的模块化和可重用性。

5.1 函数定义

使用 def 关键字定义函数。

```
def greet(name):
    """
    这个函数向指定的人打招呼。
    """
    print(" 你好,", name)

# 调用函数
greet(" 张三") # 输出: 你好, 张三
greet(" 李四") # 输出: 你好, 李四
```

- def 关键字后跟函数名 (greet) 和括号 ()。

- 括号内可以定义参数 (name)，参数是函数接收的输入。
- 函数体是缩进的代码块，包含函数要执行的语句。
- 函数可以有文档字符串 (docstring)，用 """ """ 包围，用于描述函数的功能。

5.2 参数和返回值

- **参数:** 函数可以接收零个或多个参数。参数可以是必需的或可选的 (默认参数)。

```
def add(x, y):
    """
    返回两个数的和。
    """
    return x + y

result = add(5, 3)
print(result) # 8

def power(base, exponent=2): # exponent 默认值为 2
    """
    计算 base 的 exponent 次方。
    """
    return base ** exponent

print(power(3))      # 3 的 2 次方，输出 9
print(power(3, 3))   # 3 的 3 次方，输出 27
```

- **返回值:** 函数可以使用 return 语句返回一个值。如果没有 return 语句，函数默认返回 None。

```
def get_full_name(first_name, last_name):
    """
    返回完整的姓名。
    """
    full_name = first_name + " " + last_name
```

```
    return full_name

name = get_full_name("John", "Doe")
print(name) # John Doe

def say_hello():
    print("Hello!")

return_value = say_hello()
print(return_value) # None (因为 say_hello 函数没有 return 语句)
```

5.3 作用域

变量的作用域指的是变量可以被访问的范围。Python 中主要有两种作用域:

- **局部作用域 (Local Scope):** 在函数内部定义的变量，只能在函数内部访问。
- **全局作用域 (Global Scope):** 在函数外部定义的变量，可以在整个程序中访问。

```
global_var = 10 # 全局变量

def my_function():
    local_var = 5 # 局部变量
    print(global_var) # 可以访问全局变量
    print(local_var)  # 可以访问局部变量

my_function()
print(global_var) # 可以访问全局变量
# print(local_var)  # 报错 NameError: name 'local_var' is not defined (局部变量在函数外部不可

def modify_global():
    global global_var # 使用 global 关键字声明要修改全局变量
    global_var = 20
```

```
modify_global()
print(global_var) # 全局变量被修改为 20
```

6. 类和对象

Python 是一种面向对象的编程语言。类 (Class) 是创建对象的蓝图，对象 (Object) 是类的实例。面向对象编程 (OOP) 的主要概念包括：

- **类 (Class):** 描述对象的属性和行为的模板。
- **对象 (Object):** 类的实例，具有类定义的属性和行为。
- **属性 (Attribute):** 对象的数据，也称为成员变量。
- **方法 (Method):** 对象的行为，也称为成员函数。
- **封装 (Encapsulation):** 将数据和方法捆绑在一起，隐藏内部实现细节。
- **继承 (Inheritance):** 允许一个类继承另一个类的属性和方法，实现代码重用。
- **多态 (Polymorphism):** 允许不同类的对象对同一消息做出不同的响应。

6.1 类定义

使用 `class` 关键字定义类。

```
class Dog:
    """
    表示狗的类。
    """
    def __init__(self, name, breed):
        """
        初始化 Dog 类的对象。
        """
        self.name = name
        self.breed = breed
```

```
    def bark(self):
        """
        狗叫的方法。
        """
        print(" 汪汪!")

    def describe(self):
        """
        描述狗的信息。
        """
        print(f" 这是一只 {self.breed} 狗，名字叫 {self.name}。")
```

- class 关键字后跟类名 (Dog)。
- 类体是缩进的代码块，包含类的属性和方法定义。
- __init__ 方法是构造方法 (constructor)，在创建对象时自动调用，用于初始化对象的属性。self 参数指向对象自身。
- 其他方法 (bark, describe) 定义了对象的行为。

6.2 创建对象 (实例化)

```
# 创建 Dog 类的对象 (实例)
my_dog = Dog(" 旺财", " 中华田园犬")

# 访问对象的属性
print(my_dog.name)  # 旺财
print(my_dog.breed) # 中华田园犬

# 调用对象的方法
my_dog.bark()       # 汪汪!
my_dog.describe()  # 这是一只 中华田园犬 狗，名字叫 旺财。
```

```
another_dog = Dog("Lucky", " 金毛")
another_dog.describe() # 这是一只 金毛 狗，名字叫 Lucky。
```

6.3 继承

```
class Animal:
    """
    动物基类。
    """
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(" 动物发出声音")

class Cat(Animal):
    """
    猫类，继承自 Animal 类。
    """
    def __init__(self, name, breed):
        super().__init__(name) # 调用父类的构造方法
        self.breed = breed

    def speak(self): # 重写父类的方法
        print(" 喵喵!")

    def catch_mouse(self):
        print(" 猫抓老鼠")

my_cat = Cat(" 咪咪", " 波斯猫")
print(my_cat.name)   # 咪咪 (继承自 Animal 类)
```

```
my_cat.speak()      # 喵喵! (重写了 Animal 类的方法)
my_cat.catch_mouse() # 猫抓老鼠 (Cat 类特有的方法)
```

- `class Cat(Animal)`: 表示 Cat 类继承自 Animal 类。Animal 是父类 (基类), Cat 是子类 (派生类)。
- `super().__init__(name)` 调用父类 Animal 的构造方法，初始化 name 属性。
- Cat 类重写了父类 Animal 的 speak 方法，实现了多态。

7. 错误处理

在程序运行过程中，可能会出现错误 (异常)。Python 提供了 try-except 语句来处理异常，防止程序崩溃。

```
try:
    # 可能会引发异常的代码
    num_str = input(" 请输入一个整数: ")
    num = int(num_str)
    result = 10 / num
    print(" 结果是:", result)
except ValueError:
    # 处理 ValueError 异常 (例如用户输入非数字)
    print(" 输入无效, 请输入整数。")
except ZeroDivisionError:
    # 处理 ZeroDivisionError 异常 (例如除数为零)
    print(" 除数不能为零。")
except Exception as e:
    # 处理其他类型的异常 (通用异常处理)
    print(" 发生未知错误:", e)
else:
    # 如果 try 代码块没有发生异常, 则执行 else 代码块 (可选)
    print(" 程序运行正常。")
finally:
```

```
    # 无论是否发生异常，finally 代码块都会执行 (可选)
    print(" 清理操作，无论如何都会执行。")

print(" 程序继续执行...") # 即使发生异常，程序也会继续执行到这里
```

- `try` 代码块包含可能会引发异常的代码。
- `except` 代码块用于捕获和处理特定类型的异常。可以有多个 `except` 代码块来处理不同类型的异常。
- `else` 代码块在 `try` 代码块没有发生异常时执行 (可选)。
- `finally` 代码块无论是否发生异常都会执行，通常用于清理资源 (例如关闭文件、释放连接) (可选)。
- `Exception` 是所有异常的基类，`except Exception as e:` 可以捕获所有类型的异常，并将异常对象赋值给变量 `e`，可以打印异常信息。

8. 库的安装

Python 拥有庞大的第三方库生态系统。`pip` 是 Python 的包管理工具，用于安装、升级和卸载第三方库。

8.1 使用 pip 安装库

在命令行或终端中，使用以下命令安装库:

```
pip install 库名
```

例如，安装 `requests` 库 (用于发送 HTTP 请求):

```
pip install requests
```

安装特定版本的库:

```
pip install requests==2.26.0
```

升级库到最新版本：

```
pip install --upgrade 库名
```

卸载库：

```
pip uninstall 库名
```

8.2 常用的 Python 库

- **NumPy:** 用于数值计算，提供高性能的数组和矩阵运算。
- **Pandas:** 用于数据分析和处理，提供 DataFrame 数据结构。
- **Matplotlib:** 用于数据可视化，绘制各种图表。
- **Requests:** 用于发送 HTTP 请求，进行网络编程。
- **Beautiful Soup:** 用于解析 HTML 和 XML 文档，进行网页抓取。
- **Scikit-learn:** 用于机器学习，提供各种机器学习算法和工具。
- **TensorFlow/PyTorch:** 用于深度学习，构建和训练神经网络。
- **Django/Flask:** 用于 Web 开发，构建 Web 应用程序。

9. 虚拟环境

虚拟环境是一个隔离的 Python 环境，用于管理项目依赖。它可以避免不同项目之间的库版本冲突。

9.1 创建虚拟环境

Python 自带 `venv` 模块用于创建虚拟环境。在命令行或终端中，进入项目目录，运行以下命令创建虚拟环境：

```
python -m venv venv_name  # venv_name 是虚拟环境的名称，可以自定义
```

这会在项目目录下创建一个名为 `venv_name` 的文件夹，包含虚拟环境的文件。

9.2 激活虚拟环境

- **Windows:**

```
venv_name\Scripts\activate
```

- **macOS/Linux:**

```
source venv_name/bin/activate
```

激活虚拟环境后，命令行提示符前会显示虚拟环境的名称 (`venv_name`)。

9.3 在虚拟环境中安装库

在虚拟环境激活状态下，使用 `pip install` 安装的库会安装到虚拟环境中，与全局 Python 环境隔离。

```
pip install requests  # 安装 requests 库到虚拟环境
```

9.4 退出虚拟环境

在命令行或终端中，运行以下命令退出虚拟环境：

```
deactivate
```

退出虚拟环境后，命令行提示符会恢复正常。

10. 结论

恭喜你完成了 Python 入门学习！本教材涵盖了 Python 的基础语法、数据结构、控制语句、函数、类、错误处理、库的安装和虚拟环境等核心概念。

下一步学习建议：

- **多练习:** 编写更多的 Python 代码，解决实际问题，加深理解。

- **阅读文档:** 查阅 Python 官方文档和第三方库的文档，学习更深入的知识。
- **参与项目:** 参与开源项目或自己动手做项目，将所学知识应用到实践中。
- **学习进阶主题:** 例如，模块和包、文件操作、正则表达式、多线程、网络编程、数据库编程、GUI 编程、Web 开发、数据科学、机器学习等。

Python 是一门功能强大且应用广泛的语言，希望这份教材能帮助你入门 Python 编程，开启你的编程之旅！祝你学习愉快！