



Python编程基础： 元组、字典、集合

POWERPOINT DESIGN



CONTENTS 目录

01

元组 (Tuple)

02

字典 (Dictionary)

03

集合 (Set)



Part 01

元组 (Tuple)

POWERPOINT DESIGN

元组的定义与格式



元组的格式与定义

元组是一种不可变的序列，使用小括号()定义。

例如：`tua = (1, 2, 3)`。若元组中只有一个元素，定义时需在元素后加逗号，如`tua = (1,)`。

元组中的元素可以是不同类型的数据，它们之间用逗号隔开。

定义空元组时，不需要在小括号内加逗号
如`empty_tuple = ()`。



元组的定义与格式



示例演示：

元组定义示例

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

```
print(tua) # 输出: (1, 2, 3)
```

单元素元组定义示例

```
tua_single = (4,)
```

```
print(tua_single) # 输出: (4,)
```

空元组定义示例

```
empty_tuple = ()
```

```
print(empty_tuple) # 输出: ()
```



元组的定义与格式



元组与列表的区别

元组与列表的主要区别在于元组是不可变的，而列表是可变的。这意味着一旦元组被创建，其内容不能被修改，而列表的内容可以被添加、删除或修改。

由于元组的不可变性，它们在某些情况下可以提供更好的数据安全性，因为它们不能被意外修改。



元组的定义与格式



示例演示：

列表和元组的可变性对比

```
my_list = [1, 2, 3]
```

```
my_list.append(4) # 列表可以添加元素
```

```
print(my_list) # 输出： [1, 2, 3, 4]
```

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

```
my_tuple.append(4) # 元组不支持append方法
```

```
print(my_tuple) # 输出:(1, 2, 3)
```



元组的定义与格式



元组的常见操作

元组支持查询操作，如count()、len()和index()，以及切片操作，与列表类似。

这些操作使得我们可以方便地获取元组中的元素信息。



元组的定义与格式



示例演示：

元组的常见操作示例

```
my_tuple = (1, 2, 3, 2, 4)
```

计算元素出现次数

```
count_2 = my_tuple.count(2)
```

```
print(count_2) # 输出： 2
```

获取元组长度

```
length = len(my_tuple)
```

```
print(length) # 输出： 5
```

获取元素索引

```
index_2 = my_tuple.index(2)
```

```
print(index_2) # 输出： 1
```

元组切片

```
sliced_tuple = my_tuple[1:3]
```

```
print(sliced_tuple) # 输出：
```

```
(2, 3)
```



Part 02

字典 (Dictionary)

POWERPOINT DESIGN

字典的定义与常见操作

01

字典格式

Python的字典（dictionary）是一种内置的数据类型，它用于存储键值对（key-value pairs）。

字典是无序的、可变的，并且不允许有重复的键。

每个键都与一个值关联，你可以通过键访问对应的值。

例如：`dict = {'key1': 'value1', 'key2': 'value2'}`。

注意点：键名不能重复，如果重复，后面的键值对会覆盖前面的。值可以重复。

字典的定义与常见操作

02

字典的常规操作

2.1 查看元素

1.使用 dict[key]

这种方式直接访问字典中的值。如果提供的键存在于字典中，则返回对应的值；如果键不存在，则会引发一个 `KeyError` 异常。

2.使用 get() 方法

`dict.get(key[, default])`: 这个方法尝试获取指定键的值。

如果键存在，则返回对应的值；

如果键不存在，则返回 `None` 或者你提供的默认值（如果指定了的话）。

优点：不会抛出异常，适合在不确定键是否存在的时候使用。



字典的定义与常见操作

02

字典的常规操作

2.2 修改和增加元素

`dict[key] = value`: 如果你给已存在的键赋新值, 这将更新对应的值;
如果你给一个新键赋值, 这将在字典中添加一个新的键值对。

使用场景: 当需要更新现有键的值或向字典中添加新的键值对时使用。

字典的定义与常见操作

02

字典的常规操作

2.3 删除元素

1.del 语句

`del dict[key]`: 这个语句用于删除指定键的项。

如果键不存在，则会引发一个`KeyError`异常。

2.pop(key[, default]) 方法

`dict.pop(key)`: 这个方法移除并返回指定键的值。

如果键不存在，且没有提供默认值，则抛出`KeyError`异常；

如果提供了默认值并且键不存在，则返回默认值。



字典的定义与常见操作

02

字典的常规操作

2.3 删除元素

3.clear() 方法

`dict.clear()`: 这个方法将字典中的所有项移除, 使字典变成空的。



字典的定义与常见操作

02

字典的常规操作

创建一个字典

```
my_dict = {'name': 'Alice', 'age': 25, 'city': 'Beijing'}
```

查看元素

```
print("原始字典:", my_dict)
```

使用dict[key] 查看元素

```
print("名字:", my_dict['name'])
```

使用get() 方法查看元素，如果键不存在不会报错，返回None或指定的默认值

```
print("城市:", my_dict.get('city'))
```

```
print("职业 (使用 get() 默认值):", my_dict.get('job', '未知'))
```

增加元素

```
my_dict['email'] = 'alice@example.com'
```

```
print("添加邮箱后:", my_dict)
```

修改元素

```
my_dict['age'] = 26
```

```
print("修改年龄后:", my_dict)
```



字典的定义与常见操作

02

字典的常规操作

删除元素

```
del my_dict['city']
```

```
print("删除城市后:", my_dict)
```

使用 pop() 删除元素并获取被删除元素的值

```
removed_value = my_dict.pop('email')
```

```
print("删除邮箱后:", my_dict)
```

清空字典

```
my_dict.clear()
```

```
print("清空字典后:", my_dict)
```

添加重复键名（后面的键值对会覆盖前面的）

```
my_dict = {'name': 'Bob', 'age': 30, 'name': 'Charlie'} # 'name' 的值会被 'Charlie' 覆盖
```

```
print("添加重复键名后的字典:", my_dict)
```

再次增加元素

```
my_dict['profession'] = 'Engineer'
```

```
print("最终字典:", my_dict)
```



字典的定义与常见操作

03

字典的高级操作

1.len(dict)

返回字典中键值对的数量。

2.keys()

返回一个包含字典所有键的视图对象，该视图对象可以转换为列表。

3.values()

返回一个包含字典所有值的视图对象，该视图对象也可以转换为列表。

4.items()

返回一个包含字典所有键值对（元组形式）的视图对象，同样可以转换为列表。

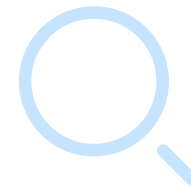


字典的定义与常见操作

03

字典的高级操作

```
my_dict = {'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}  
length = len(my_dict)  
print(length) # 输出: 3  
keys = my_dict.keys()  
print(list(keys)) # 输出: ['key1', 'key2', 'key3']  
values = my_dict.values()  
print(list(values)) # 输出: ['value1', 'value2', 'value3']  
items = my_dict.items()  
print(list(items)) # 输出: [('key1', 'value1'), ('key2', 'value2'), ('key3', 'value3')]
```



Part 03

集合 (Set)

POWERPOINT DESIGN



集合的定义与操作



01

集合格式及注意点

集合是一个无序且元素唯一的容器，使用花括号`{}`。例如：`set1 = {1, 2, 3}`。

集合中的元素是唯一的，不能有重复元素，这使得集合非常适合用于去重操作。

空集合的定义为`set()`。

02

集合的常见操作

集合不支持根据索引修改或索引元素，但可以添加元素`add()`和`update()`，删除元素`remove()`和`pop()`，以及`discard()`，后者在元素不存在时不会报错。

03

集合的交集与并集

交集使用`&`操作符，如果两个集合没有交集，则返回空集合`set()`。

并集使用`|`操作符，将两个集合中的所有元素合并，重复的元素只计算一次，因为集合中的元素是唯一的。

字典的定义与常见操作

01

集合格式及注意点示例

集合的基本操作示例

```
my_set = {1, 2, 3}
```

元素的唯一性

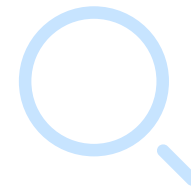
```
my_set.add(1) # 不会重复添加
```

```
print(my_set) # 输出: {1, 2, 3}
```

定义空集合

```
empty_set = set() # 正确的方式创建空集合
```

```
print(type(empty_set)) # 输出: <class 'set'>
```



字典的定义与常见操作

02

集合的常见操作示例

集合的常见操作示例

```
my_set = {1, 2, 3}
```

添加元素

```
my_set.add(4)
```

```
print(my_set) # 输出: {1, 2, 3, 4}
```

删除元素

```
my_set.remove(2)
```

```
print(my_set) # 输出: {1, 3, 4}
```

使用pop()方法

```
popped_element = my_set.pop()
```

```
print(popped_element) # 输出可能是1、3或4中的任意一个
```

```
print(my_set) # 输出: {1, 3, 4} 或 {1, 4, 3} 或 {3, 4, 1}, 取决于pop()删除的是哪个元素
```

使用discard()方法

```
my_set.discard(5) # 元素不存在时不会报错
```

```
print(my_set) # 输出: {1, 3, 4}
```

```
a=my_set[0]
```

```
print(a)
```



字典的定义与常见操作

03

集合的交集与并集示例

#定义两个集合

```
set_a = {1, 2, 3}
```

```
set_b = {2, 3, 4}
```

交集

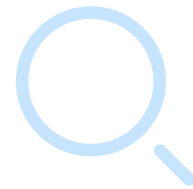
```
intersection = set_a & set_b
```

```
print(intersection) # 输出: {2, 3}
```

并集

```
union = set_a | set_b
```

```
print(union) # 输出: {1, 2, 3, 4}
```





谢谢大家

主讲人

