

第2章 变量和简单的数据类型

变量和字符串

变量指向一个特定的值，是一个可被赋值的标签。字符串就是一系列用单引号或双引号包裹的字符。

演示代码 2-1

```
name = 'iTuring'          # 字符串变量
print(f'hello {name}')    # 输出 f 字符串
```

运行结果
hello iTuring

在字符串中添加空白

- \t: 在字符串中表示制表符。
- \n: 在字符串中表示换行符。

演示代码 2-2

```
print('Python Rust')
print()
print('Python\tRust') # 使用制表符
print()
print('Python\nRust') # 使用换行符
```

运行结果
Python Rust
Python Rust
Python Rust

字符串的常用方法

- title(): 将每个单词的首字母都改为大写。
- upper()/lower(): 将字符串全部改为大写 / 小写。
- rstrip()/lstrip()/strip(): 移除字符串开头 / 末尾 / 两边的空白。
- removeprefix(val)/removesuffix(val): 移除字符串中指定的前缀 / 后缀。

演示代码 2-3

```
# 在一行中给多个变量赋值（见 2.4.5 节）
msg1, msg2 = 'hello world', ' iTuring'
print(f'{msg1.title()}\n{msg1.upper()}')
print(f'{msg2}\n{msg2.strip()}')
print(f'fname.txt'.removesuffix('.txt'))
```

运行结果
Hello World
HELLO WORLD
iTuring
iTuring
fname

整数值、浮点数（小数）值、布尔值

演示代码 2-4

```
print(1, 1_000_000, 1.5)
print(True, False) # 布尔值
print(1 + 1, 1.0 + 1.0, 1 + 1.0)
# 乘法、除法、整除法（向下取整）和求模
print(3 ** 2, 3 / 2, 3 // 2, 5 % 2)
```

运行结果
1 1000000 1.5
True False
2 2.0 2.0
9 1.5 1 1

第5章 if 语句

if 语句将通过检查条件测试语句，以选定执行对应的逻辑。

条件测试

- ==、!=: 用于判断两个值是否相等 / 不相等。
- <、>、<=、>: 用于数值间的大小比较。
- and、or、not 关键字: 用于布尔运算，分别表示和 / 或 / 非。
- in 关键字: 用于检查特定值是否在集合中。

演示代码 5-1

```
condition1 = (100 == 100)      # True
condition2 = (100 != 100)      # False
condition3 = (100 < 200)       # True
condition4 = (100 >= 200)      # False
condition5 = ('a' in ['a'])    # True
condition6 = ('a' not in ['a']) # False
condition7 = condition1 or condition2 # True
condition8 = condition1 and condition2 # False
```

if 语句

依次检查每个条件测试，选择执行首个为真或 else 中的子块代码。

- elif: 可有零个或多个，当 if 的条件为假时，依次检查处理。
- else: 可有零个或一个，处理 if 和 elif 的条件均为假的情况。

演示代码 5-2

```
age, vip = 12, False
if 0 <= age < 4 and vip:
    ticket_price = 0
elif age < 18:
    ticket_price = 25
elif age < 65:
    ticket_price = 40
else:
    ticket_price = 20
print(f'Ticket price is {ticket_price}.')
```

运行结果
Ticket price is 25.

第3章 列表简介

列表是由一系列按特定顺序排列的元素组成的序列，使用方括号（包裹）表示，可以使用索引来访问其中的元素。

列表的创建与元素访问

演示代码 3-1

```
bikes = ['trek', 'redline', 'giant']
print(bikes)
print(bikes[0]) # 索引从 0 开始，第一个元素的索引是 0
print(bikes[2], bikes[-1])
```

运行结果
['trek', 'redline', 'redline']
trek
giant giant

修改、添加和删除元素

- lst[idx] = val: 修改列表中指定索引的元素。

演示代码 3-2

```
bikes = ['trek']
bikes[0] = 'giant'
print(bikes)
```

运行结果
['giant']

- lst.append(val): 在列表的末尾添加新元素。
- lst.insert(idx, val): 在列表的指定位置上添加新元素。

演示代码 3-3

```
bikes = []
bikes.append('redline')
bikes.insert(0, 'trek')
print(bikes)
```

运行结果
['trek', 'redline']

- lst.pop([idx]) -> val: 删除并返回列表中指定索引（默认为末尾）的元素。

演示代码 3-4

```
bikes = ['giant', 'trek', 'redline']
bikes.pop(0)      # 移除 'giant' 元素
print(bikes.pop()) # 移除 'trek' 元素并输出
print(bikes)
```

运行结果
redline
['trek']

- del lst[idx]: 删除列表中指定索引的元素。
- lst.remove(val): 删除列表中匹配到的第一个指定元素。

演示代码 3-5

```
bikes = ['giant', 'redline', 'trek']
del bikes[0]      # 移除 'giant' 元素
bikes.remove('trek') # 移除 'trek' 元素
print(bikes)
```

运行结果
['redline']

列表相关的常用方法

- lst.sort(): 永久修改原列表，对其中的元素进行排序。
- lst.reverse(): 永久修改原列表，对其中的元素进行翻转。
- sorted(lst) -> lst: 返回排序后的列表的副本。
- len(lst) -> num: 获取列表的元素个数。

演示代码 3-6

```
nums = [9, 6, 1, 4, 2]
print(len(nums))
print(sorted(nums))
print(nums) # 原列表不变
# 列表排序（默认正序）
nums.sort()
print(nums)
# 列表排序（指定倒序）
nums.sort(reverse=True)
print(nums)
nums.reverse() # 翻转列表
print(nums)
```

运行结果
5
[1, 2, 4, 6, 9]
[9, 6, 1, 4, 2]
[1, 2, 4, 6, 9]
[9, 6, 4, 2, 1]
[1, 2, 4, 6, 9]

第4章 操作列表

循环遍历列表

演示代码 4-1

```
bikes = ['trek', 'redline']
for bike in bikes:
    print(bike.title())
```

运行结果
Trek
Redline

创建数值列表和列表推导式

- range([start,] end [,step]): 生成可迭代的数值列表的表示。
- max(lst)/min(lst)/sum(lst): 对数值列表执行简单的统计计算。

演示代码 4-2

```
print(range(2))
print(list(range(2)))

squares = []
for value in range(2, 5):
    squares.append(value ** 2)

squares_comp = [value ** 2 for value in range(2, 5)]
print(squares == squares_comp) # 推导式会创建相同的元素
print(squares)
print(min(squares), max(squares), sum(squares))
```

运行结果
range(0, 2)
[0, 1]
True
[4, 9, 16]
4 16 29

关于列表推导式的相关内容，详见 4.3.4 节。

创建列表切片

列表切片将返回一个原列表的副本，我们可以利用这个特性来复制列表。

演示代码 4-3

```
bikes = ['trek', 'redline', 'giant']
print(bikes[1:])
print(bikes[:-1])
print(bikes[0:1])
bikes_copy = bikes[:]
# 对副本的操作，不影响原列表
bikes_copy.reverse()
print(bikes)
print(bikes_copy)
```

运行结果
['redline', 'giant']
['trek', 'redline']
['trek']
['trek', 'redline', 'giant']
['giant', 'redline', 'trek']

元组的定义和遍历

元组是不可变的列表，使用圆括号（包裹）表示。

演示代码 4-4

```
dimensions = (200, 50)
for value in dimensions:
    print(value)

# 元组无法修改，但变量可以被重新赋值
dimensions = (300, 100)
print(dimensions)
dimensions = 100
print(dimensions)
```

运行结果
200
50
(300, 100)
100

常用内置函数

print()	打印输出
help()	用于查看函数或模块用途的详细说明
list()	将一个可迭代对象转换成列表
tuple()	将一个可迭代对象转换成元组
set()	将一个可迭代对象转换成集合
dict()	用于创建一个新字典
sorted()	将一个序列排序，返回排序后的序列
reversed()	将一个序列翻转，返回翻转序列的迭代器
range()	用于生成可迭代的数值列表的表示
eval()	执行字符串类型的代码，并返回最终结果
min()、max()、sum()	返回最小值、最大值、总和
len()	用于获取一个序列或集合的长度
zip()	返回一个迭代器，其中包含将多个序列中同一位置的元素压缩成的一个个元组



扫码领取  
随书代码资料

## 第6章 字典

字典是由一系列键值对组成的集合，使用花括号（包裹）表示。键值之间互相关联，可以使用键来访问其关联值。

### 字典的创建与访问

#### 演示代码 6-1

```
alien = {'color': 'green', 'points': 5}
print(alien['color'], alien.get('color'))
```

#### 运行结果

```
green green
```

### 修改、添加和删除键值

- `d[key] = val`: 如果键存在，则修改对应的值，否则添加新键值对。

#### 演示代码 6-2

```
alien = {'color': 'green', 'points': 5}
alien['color'] = 'red'
alien['position'] = (0, 25)
print(alien)
```

#### 运行结果

```
{'color': 'red', 'points': 5, 'position': (0, 25)}
```

- `del d[key]`: 根据键删除指定的键值对。
- `d.pop(key) -> val`: 根据键删除指定的键值对，返回键对应的值。

#### 演示代码 6-3

```
d = {'a': 1, 'b': 2, 'c': 3}
del d['a']
print(d.pop('c'))
print(d)
```

#### 运行结果

```
3
{'b': 2}
```

### 字典的常用方法

- `d.items()`: 返回所有键值对的元组视图。
- `d.keys()/d.values()`: 返回所有键 / 值的列表视图。
- `len(d)`: 获取字典的键值对的组数。

#### 演示代码 6-4

```
favorites = {'jen': 'python', 'edward': 'rust'}
print(len(favorites))
for name, language in favorites.items():
    print(f"{name} loves {language}.")

print(list(favorites.keys()))
print(list(favorites.values()))
```

#### 运行结果

```
2
jen loves python.
edward loves rust.
['jen', 'edward']
['python', 'rust']
```

### 字典推导式

#### 演示代码 6-5

```
squares = {x: x ** 2 for x in range(4)}
keys, vals = [0, 1, 2, 3], [0, 1, 4, 9]
squares_zip = {key: val
                for key, val in zip(keys, vals)}

print(squares == squares_zip, squares)
```

#### 运行结果

```
True {0: 0, 1: 1, 2: 4, 3: 9}
```

关于字典与字典、字典与列表嵌套的相关介绍，详见 6.4 节。

## 常用内置函数

<code>open()</code>	用于打开一个文件，创建一个文件句柄
<code>all()</code>	可迭代对象中全部是 <code>True</code> ，结果才是 <code>True</code>
<code>any()</code>	可迭代对象中有一个是 <code>True</code> ，结果就是 <code>True</code>
<code>abs(x)</code> 、 <code>pow(x)</code>	取绝对值
<code>str(x)</code>	转为字符串类型
<code>int(x)</code> 、 <code>float(x)</code> 、 <code>bool(x)</code>	转为整数、浮点数或布尔类型
<code>input()</code>	获取用户输入的内容
<code>type(obj)</code>	获取传入对象的类型

本速查地图由陶叶港（@ScrueI）整理编写。

## 第7章 用户输入和 while 循环

### 用户输入

#### 演示代码 7-1

```
name = input("Please enter your name: ")
print(f"\nHello, {name}!")
```

#### 运行结果

```
Please enter your name: iTuring
Hello, iTuring!
```

### 字符串转换其他类型

- `int(string)`: 将字符串转换为整数值。
- `float(string)`: 将字符串转换为浮点数（小数）值。

#### 演示代码 7-2

```
age = input("How old are you? ")
age = int(age)
print(type(age), type(int(age)))
pi = float(input("What's the value of pi? "))
```

### while 循环

- 不断运行 `while` 语句块中的代码，直到给定的条件为假。
- `break` 关键字：退出整个循环语句块的执行。
- `continue` 关键字：跳过当前循环中余下代码的执行。

#### 演示代码 7-3

```
current_number = 0
while True:
    current_number += 1
    if current_number > 5:
        break
    if current_number % 2 == 0:
        continue
    print(current_number)
```

#### 运行结果

```
1
3
5
```

## 第10章 文件和异常

### 读写文件

#### 演示代码 10-1

```
from pathlib import Path
path = Path('msg.txt')
path.write_text('Hello iTuring!\nHappy coding!')
```

```
contents = path.read_text()
print(len(contents.splitlines()))
print(contents)
```

#### 运行结果

```
2
Hello iTuring!
Happy coding!
```

### 异常

异常是 Python 创建的特殊对象，用于管理程序运行时出现的错误。

#### 演示代码 10-2

```
try:
    num = int(input("Type a number: "))
    answer = 5 + num
except ValueError:
    print("You must type a number!")
else:
    print(answer)
finally:
    print("Exit.")
```

#### 运行结果 1

```
Type a number: a
You must type a number!
Exit.
```

#### 运行结果 2

```
Type a number: 1
6
Exit.
```

### 序列化（json 库）

#### 演示代码 10-3

```
import json

numbers = [2, 3, 5, 7, 11, 13]
contents = json.dumps(numbers)
numbers_load = json.loads(contents)
print(contents, numbers_load == numbers)
```

#### 运行结果

```
[2, 3, 5, 7, 11, 13] True
```

## 第8章 函数

函数是带名字的代码块，通过使用函数，可以避免编写重复的代码。我们可以根据函数的形参，来传入位置实参或关键字实参，使程序表现出类似但不尽相同的行为。

### 函数定义

#### 演示代码 8-1

```
# 形参 color 有默认值
def describe_pet(animal, name, color='yellow'):
    """ 显示宠物的信息 """
    print(f"My {animal}'s name is {name}, color is {color}.")
```

```
describe_pet('hamster', 'harry')
describe_pet(name='willie', color='black', animal='dog')
```

#### 运行结果

```
My hamster's name is harry, color is yellow.
My dog's name is willie, color is black.
```

### 函数的返回值

#### 演示代码 8-2

```
def add_if_same_or_sub(x, y):
    if x == y:
        return x + y
    return x - y
```

#### 运行结果

```
3 4
```

```
val1 = add_if_same_or_sub(5, 2)
val2 = add_if_same_or_sub(2, 2)
print(val1, val2)
```

### 不定长参数

- `*args`: 以一个星号为前缀，将接受一个包含多余位置参数的元组。
- `**kwargs`: 以两个星号为前缀，将接受一个包含多余关键字参数的字典。

#### 演示代码 8-3

```
def func(*args, **kwargs):
    print(f"args: {args}\nkwargs: {kwargs}")
```

```
func(1, 2, k='v')
```

#### 运行结果

```
args: (1, 2)
kwargs: {'k': 'v'}
```

关于模块的相关内容，详见 8.6 节、9.4 节及 10.1 节等。

## 第9章 类

类是一种抽象的数据类型，代表某种实体的抽象概念，描述了所有该类的对象共享的属性和行为，对象是类的实例。

#### 演示代码 9-1

```
class Car:
    def __init__(self, model, energy_capacity=100):
        """ 初始化类的属性 """
        self.model = model
        self.energy_capacity = energy_capacity

    def show_dashboard(self):
        print(f'Energy remain: {self.energy_capacity}%')

    def drive(self, distance_km):
        # 修改类的属性
        self.energy_capacity -= distance_km // 5
```

#### 运行结果

```
Energy remain: 90%
Battery is safe!
Energy remain: 100%
```

```
class ElectricCar(Car):
    def __init__(self, model):
        """ 初始化子类的属性，子类继承了父类的方法和属性，
        我们可以调用父类的初始化方法来初始化父类的属性 """
        super().__init__(model)
        self.battery_status = 'safe'

    def check_battery_safe(self):
        print(f'Battery is {self.battery_status}!')
```

```
mycar = ElectricCar('Tesla-ModelX')
mycar.drive(50), mycar.show_dashboard()
mycar.check_battery_safe()
mycar.recharge(), mycar.show_dashboard()
```