考点

面向Python初学者

- 基本语法
- 函数
- 文件操作

基本语法

变量与赋值

基本概念

赋值、变量、表达式、运算

代码

```
x = 1 # 1 赋值给变量x
y = x * 2 + 1  # 将表达式2x+1的值赋值给y

logo = "Snake" # 将字符串"Snake"赋值给logo
logo = "Python" # 覆盖原值
```

知识点

- `=`标识赋值，不是判断相等；左侧是标识符，右侧是值或表达式
- 变量必须用合法的标识符；合法的标识符都是变量。Python没有真正的常量

类型

基本概念

- 内建类型：
    1. 整数型、Boole型、浮点型、字符串
    2. 列表、数组、字典
- 类型注释
- 可变变量、不可变变量、副作用

代码

```
year: int = 2025 # 类型注释
pi: float = 3.14

name: str = "Trump"

# 列表，字符串列表
presidents: list = ["Trump", "Biden"]
presidents: list[str] = ["Trump", "Biden"]

# 列表嵌套
tree: list = ["root", ["branch 1", "branch 2"], ["branch3", ["leaf 1", "leaf
2"]]]

coordinates: tuple = (0, 0, 0)

student: dict = {"name": "Einstein", "nation":("America","Gemany"),
"gender":True}
```

```
# 利用类名创建
empty = list()

# 类型转换
pi = float("3.14")
price = int(input("input the price of the macbook: $"))

presidents = tuple(["Trump", "Biden"]) # list -> tuple
presidents = list(("Trump", "Biden"))  # tuple -> list
python = list("python") # str -> list
```

代码：类型判断

```
# 类型判断
print(type(year)) # int
print(type(name)) # str

print(instance(year, int)) # True
print(instance(name, int)) # False
```

```
# 表达式/基本运算
3 + 4
985 - 211
4 * 100
5 / 8 # 0.625
5 // 8 # 0

"come" + " " + "on" == "come on"
"love" * 3 == "lovelovelove"
```

知识点

- 可变：列表、字典；不可变：整数型、Boole型、浮点型、字符串
- Boole型是特殊的整数型；
- 列表中的元素没有类型限制；甚至可以嵌套
- 元组和列表表示数据时完全一致，但元组不可变，列表可变；
- 可变与不可变变量的基本区别：
  1. 前者有*副作用方法*，后者没有
  2. 在id不变的情况下，前者可改变值，后者不可改变
- 类型注释不影响程序执行（程序反省时可以利用这些信息）
- 可利用类名进行创建和类型转换

索引与切片

概念

索引、索引赋值、键索引

代码

```python
# 字符串索引
text = "Python"
print(text[0])      # 'P' - 正向索引
print(text[-1])     # 'n' - 负向索引
print(text[1:4])    # 'yth' - 切片 [start:end)
print(text[:3])     # 'Pyt' - 从头开始
print(text[3:])     # 'hon' - 到末尾
print(text[::2])    # 'Pto' - 步长为2
print(text[::-1])   # 'nohtyP' - 反转字符串
# 列表索引（元组等同）
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(numbers[2:7:2])  # [2, 4, 6] - 从2到7，步长2
print(numbers[5:])     # [5, 6, 7, 8, 9] - 从5开始
print(numbers[:5])     # [0, 1, 2, 3, 4] - 到5结束

# 字典键索引
person = {"name": "Trump", "age": 80, "city": "New York"}
print(person["name"])

print(person["height"])  # KeyError，键不存在
print(person.get("height", 180))  # 180，使用默认值
```

代码：索引赋值

```python
# 列表索引赋值
lst = [1, 2, 3, 4]
lst[1] = 20  # 修改第二个元素
print(lst)   # [1, 20, 3, 4]

# 切片赋值
lst[1:3] = [200, 300]  # 替换切片部分
# lst[1:3] = 200, 300  # 等价
print(lst)   # [1, 200, 300, 4]

# 字典索引赋值
person["height"] = 180  # height 不存在就自动创建
```

```
# 扩展切片赋值
lst[1:1] = [500, 600]  # 在索引1处插入
print(lst)   # [1, 500, 600, 200, 300, 4]

# 删除元素
del lst[2]  # 删除索引2的元素
print(lst)  # [1, 500, 200, 300, 4]
```

索引赋值与修改

```
# 列表索引赋值
lst = [1, 2, 3, 4]
lst[1] = 20  # 修改第二个元素
print(lst)   # [1, 20, 3, 4]

# 切片赋值
lst[1:3] = [200, 300]  # 替换切片部分
print(lst)   # [1, 200, 300, 4]

# 扩展切片赋值
lst[1:1] = [500, 600]  # 在索引1处插入
print(lst)   # [1, 500, 600, 200, 300, 4]

# 删除元素
del lst[2]  # 删除索引2的元素
print(lst)  # [1, 500, 200, 300, 4]
```

知识点

- 索引从0开始；切片左闭右开
- -1表示从最后一位开始倒数
- 列表、元组、字符串索引完全等同
- 字典必须用键名索引，无切片操作
- 列表、字典可做索引赋值；列表可做切片赋值；元组是不可变变量，不能做索引赋值
- 字典索引赋值会自动创建

控制语句

基本概念

- 条件语句、循环语句
- Boole运算

关键字

```
if elif else
and or not
for-in while break continue
```

条件语句

```
price = 1000

if score >= 1000:
    level = "奢侈"
elif score >= 500:
    level = "昂贵"
elif score >= 100:
    level = "中等"
else:
    level = "便宜"

print(f"分数{price}对应等级`{level}`")

# 链式比较
age = 25
if 18 <= age <= 60:
    print("成年人")

# 布尔运算
is_student = True
has_id = False
if is_student and not has_id:
    print("需要办理学生证")
```

循环语句

```python
# for循环
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# 带索引的for循环
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")

# 字典的键值对循环
fruits = {"apple": 100, "banana": 200, "cherry": 800}
print("fruit shop:")
for fruit, price in fruits.item():
    print(f"{apple}: {price} $/kg")

# range循环
for i in range(5):       # 0到4, 等价于range(0,5,1)
    print(i)

for i in range(2, 10, 2):  # 2,4,6,8
    print(i)

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

# 循环控制
for i in range(10):
    if i == 3:
        print("continue the loop directly, instead of print i.")
        continue  # 跳过本次循环剩余代码
    if i == 7:
        print("end the loop!")
        break     # 终止整个循环
    print(i)
else:
    # 循环正常结束时执行（非break退出）
    print("循环正常结束")
```

知识点

- 布尔运算
- range整数循环

- 列表、字典遍历
- Python没有case语句、没有do-while语句

函数

关键字

```
def return lambda
```

基本概念

- 函数定义与调用
- 函数返回值
- 函数参数：
    1. 形式参数：普通参数、带默认值参数、可变位置参数、可变关键字参数
    2. 实际参数：位置参数、关键字参数
- 函数参数的类型注释
- 局部变量、全局变量、作用域

函数定义与调用

代码

```python
# 基本函数
def greet(name):
    """向某人问好"""
    return f"Hello, {name}!"

def greet(name:str)->str:
    """向某人问好;
    参数类型注释
    """
    return f"Hello, {name}!"

# 带默认参数的函数
def calculate_area(length, width=1):
    """计算矩形面积"""
    return length * width

# 可变位置参数
def sum_numbers(*args):
    """计算任意数量数字的和"""
    total = 0
    for num in args:
        total += num
    return total

# 可变关键字参数
def create_person(**kwargs):
    """创建人员信息字典"""
    person = {"id": 1001}
    person.update(kwargs)
    return person

# 混合参数类型
def complex_function(pos1, pos2, /, standard, *, kwonly1, kwonly2=10):
    """
    参数传递规则:
    - / 前的参数必须按位置传递
    - / 后的参数可以按位置或关键字传递
    - * 后的参数必须按关键字传递
    """
    return pos1 + pos2 + standard + kwonly1 + kwonly2
```

高阶函数

- 高阶函数：以函数作为参数，或返回值
- 匿名函数

- 装饰器

代码

```
# 返回多个值（实际上是返回元组）
def get_min_max(numbers):
    """返回最小值和最大值"""
    return min(numbers), max(numbers)

# 函数作为参数（高阶函数）
def apply_operation(func, x, y):
    """应用指定的操作到x和y"""
    return func(x, y)

result = apply_operation(lambda a, b: a * b, 5, 3)
print(result)  # 15

# 闭包函数
def make_multiplier(factor):
    """创建乘法器函数"""
    def multiplier(x):
        # 局部函数，可使用factor参数
        return x * factor
    return multiplier

double = make_multiplier(2)
triple = make_multiplier(3)
print(double(5))  # 10
print(triple(5))  # 15
```

代码：装饰器

```python
import time

def timer(func):
    """计算函数执行时间的装饰器
    """
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        print(f"{func.__name__} 执行时间: {end_time - start_time:.4f}秒")
        return result
    return wrapper

@timer
def slow_function():
    """模拟耗时操作
    等价于 slow_function = timer(slow_function)
    """
    time.sleep(1)
    return "完成"

# 使用装饰器
result = slow_function()
```

知识点

- 函数定义相当于对变量赋值；函数调用相当于变量运算
- 函数内定义的变量是局部变量作用于仅在函数内部
- 函数可以使用函数体外的任何变量；函数可以操作函数体外的可变变量和用global标注的全局变量
- 函数可以作为函数的参数和返回值

文件操作

关键字、内建函数

```
open
with-as
```

基本概念

- 文件、路径、目录/文件夹

- 文本文件、二进制文件（图片、视频、pdf文档）
- 文件对象（指针）、文件读写
- 打开文件模式：

模式	描述	文件不存在
'r'	只读（默认）	抛出错误
'w'	写入（覆盖）	创建文件
'a'	追加	创建文件
'x'	创建（独占）	创建文件，存在则失败
'b'	二进制模式	-
't'	文本模式（默认）	-
'+'	读写模式	-

- 当前工作目录、根目录、同级目录、子目录、父目录
- 绝对路径、相对路径

代码：基本文件操作

```python
# 1. 读取文件example.txt (位于当前工作目录下)
with open('example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)

# 2. 逐行读取
with open('example.txt') as file:
    for line in file:
        print(line.strip())  # strip()移除换行符

# 3. 读取所有行到列表
with open('example.txt') as file:
    lines = file.readlines()
    print(lines)

# 4. 写入文件
title = "标题"
data = ["第一行\n", "第二行\n", "第三行\n"]
with open('output.txt', 'w', encoding='utf-8') as file:
    file.write(title)
    file.writelines(data)


n = "01"
with open('output.txt', 'w+') as file:
    file.write(n)
    file.seek(-1, 0)
    n = file.read()
    file.write(str(int(n)+1))

# 5. 追加内容
with open('output.txt', 'a', encoding='utf-8') as file:
    file.write("追加的内容\n")

# 6. 读写二进制文件
with open('image.jpg', 'rb') as file:
    binary_data = file.read()
    # 处理二进制数据...
```

知识点

- 操作file仅改变输入流；执行file.close()后正式完成对文件的写入；
- 退出 with语句 会自动执行file.close()
- 读写操作会改变文件指针位置
- 当前目录用.表示；父目录用..表示，以此类推

目录操作

代码：使用os模块

```
import os

# 获取当前工作目录
current_dir = os.getcwd()
print(f"当前目录: {current_dir}")

# 列出目录内容
items = os.listdir('.')
print(f"目录内容: {items}")

# 创建目录
os.makedirs('new_folder/subfolder', exist_ok=True)

# 删除文件
if os.path.exists('temp.txt'):
    os.remove('temp.txt')

# 删除空目录
if os.path.exists('empty_dir'):
    os.rmdir('empty_dir')

# 递归删除目录
import shutil
if os.path.exists('dir_to_remove'):
    shutil.rmtree('dir_to_remove')

# 检查路径类型
path = 'example.txt'
print(f"是否存在: {os.path.exists(path)}")
print(f"是文件: {os.path.isfile(path)}")
print(f"是目录: {os.path.isdir(path)}")
print(f"绝对路径: {os.path.abspath(path)}")
```

代码：pathlib库简介（现代文件路径操作）

```python
from pathlib import Path

# 创建Path对象
current_dir = Path.cwd() # 等价于 Path('.')
home_dir = Path.home()
file_path = Path('data/sample.txt')

# 路径操作
print(f"当前目录: {current_dir.absolute()}")
print(f"父目录: {file_path.parent}")
print(f"文件名: {file_path.name}")
print(f"后缀名: {file_path.suffix}")
print(f"无后缀文件名: {file_path.stem}")

# 路径拼接
new_path = current_dir / 'data' / 'output.txt'
print(f"新路径: {new_path}")

# 创建目录
data_dir = Path('data/results')
data_dir.mkdir(parents=True, exist_ok=True)

# 文件操作
if file_path.exists():
    content = file_path.read_text(encoding='utf-8')
    print(f"文件内容: {content[:20]}...")
else:
    file_path.parent.mkdir(parents=True, exist_ok=True)  # 强制新建
    file_path.write_text("这是新文件的内容\n", encoding='utf-8')

# 遍历目录
for item in current_dir.iterdir():
    if item.is_file():
        print(f"文件: {item.name}")
    elif item.is_dir():
        print(f"目录: {item.name}")

# 模式匹配
py_files = list(current_dir.glob('*.py'))
print(f"Python文件: {[f.name for f in py_files]}")
```